On Searching Maximum Directed (k, ℓ)-Plex

Shuohao Gao[§], Kaiqiang Yu[‡], Shengxin Liu[§], Cheng Long[‡], Zelong Qiu[§]

[§]*Harbin Institute of Technology, Shenzhen, China*, [‡]*Nanyang Technological University, Singapore*

{200111201@stu., sxliu@}hit.edu.cn, {kaiqiang002@e., c.long@}ntu.edu.sg, qiuzelong.qzl@gmail.com

***Abstract*—Finding cohesive subgraphs from a directed graph is a fundamental approach to analyze directed graph data. We consider a new model called directed (k, ℓ)-plex for a cohesive directed subgraph, which is generalized from the concept of k-plex that is only applicable to undirected graphs. Directed (k, ℓ)-plex has the connection requirements on both inbound and outbound directions of each vertex inside, i.e., each vertex disconnects at most k vertices and is meanwhile not pointed to by at most ℓ vertices. In this paper, we study the maximum directed (k, ℓ)-plex search problem which finds a directed (k, ℓ)-plex with the most vertices. We formally prove the NP-hardness of the problem. We then design a heuristic algorithm called `DPHeuris`, which finds a directed (k, ℓ)-plex with the size close to the maximum one and runs practically fast in polynomial time. Furthermore, we propose a branch-and-bound algorithm called `DPBB` to find the exact maximum directed (k, ℓ)-plex and develop effective graph reduction strategies for boosting the empirical performance. Finally, we conduct extensive experiments on real directed graphs. The experimental results show that (1) our heuristic method can quickly find a near-optimal solution and (2) our branch-and-bound method runs up to six orders of magnitude faster than other baselines.**

***Index Terms*—Cohesive subgraph mining, directed graphs, directed (k, ℓ)-plex**

I. Introduction

Directed graphs are widely used to capture the relationships among entities in many real applications. Some examples include social media, e.g., Twitter, where the "following" relationships between pairs of users can be modeled as directed edges [26], and Web networks where the hyperlinks between web pages can also be captured as directed edges [1]. To analyze these graphs, cohesive subgraph mining is a fundamental approach, which has been widely studied in the recent years [14], [15], [17], [49]. Specifically, given a directed graph, it aims to extract dense/cohesive directed subgraphs that help to solve practical problems.

Quite a few models have been proposed for a cohesive directed subgraph, including directed clique [37], directed (k, ℓ)-core [18], [21] and directed quasi-clique [23]. All are generalized naturally from their counterparts (namely clique [7], [9], [31], [43], k-core [4], [12], [25] and quasi-clique [28], [36], [50], [51]) in undirected graphs by imposing the (dis)connection requirements on both *inbound* and *outbound* directions of each vertex inside. For example, directed clique requires that each vertex connects to and is meanwhile pointed to by all other vertices [37]; directed (k, ℓ)-core requires that each vertex connects to at least k vertices and is meanwhile pointed to by at least ℓ vertices, where k and ℓ are positive integers specified by users [21]. As a result, vertices in cohesive directed subgraphs are highly connected to and from each other.

In this paper, we consider a new cohesive subgraph model called *directed* (k, ℓ)*-plex* (DPlex) which is naturally generalized from the cohesive model of k-plex (say, a graph structure where each vertex disconnects at most k vertices including itself [38]) in undirected graphs. DPlex requires that each vertex disconnects at most k vertices and is not pointed to by at most ℓ vertices, where k and ℓ are two small positive integers specified by users. We study the problem of finding a DPlex with the largest number of vertices, called *maximum DPlex*, for the following considerations. First, while k-plex is useful in graph data analysis, it is defined for undirected graphs and ignores the direction information of edges. DPlex is a meaningful adaptation of k-plex to mine directed graphs. Second, DPlex inherits some nice properties of k-plex. Specifically, DPlex relaxes the directed clique model by allowing some disconnections for each vertex. Thus, it suits better many real application scenarios since data quality issues (e.g., noises and faults) naturally exist in the data acquisition process. Besides, DPlex satisfies the *hereditary property*, i.e., any subgraph of a DPlex is still a DPlex. This can be utilized to develop efficient algorithms for finding maximum DPlex. Finally, finding the maximum DPlex can potentially be used for various applications such as biologically relevant functional group discovery [2], [6], [8], [24], community search [16], [23], anomaly detection [41], [46], [48] and etc. We have conducted a case study of community search, which shows that the maximum DPlex performs better than other directed cohesive subgraph structures including directed core, directed truss, directed clique and directed quasi-clique (details will be presented in Section V.A).

Same as the problem of finding the maximum k-plex in undirected graphs, the problem of finding the maximum DPlex is NP-hard, as we prove in this paper. In addition, while there are quite a few algorithms proposed for finding the maximum k-plex in undirected graphs [10], [19], [27], [45], [47], [52], [53], they cannot find the maximum DPlex in directed graphs since they do not capture the direction information of edges.

In this paper, we first introduce a *heuristic* algorithm called `DPHeuris`, which returns a near-maximum DPlex with the size close to the maximum one but runs in polynomial time. Specifically, it builds the output DPlex g starting from an empty graph, then iteratively selects one vertex from the remaining based on a carefully-designed heuristic strategy and includes it to g while maintaining that g is a DPlex until this is not possible. Besides, we introduce a *branch-and-bound* algorithm called `DPBB`, which finds the exact maximum DPlex but runs in exponential time due to the NP-hardness of our problem. To boost the practical performance, we develop a few effective techniques including 1) tailored graph reduction strategies that remove unpromising vertices at a proper time, and 2) new upper bounding methods that provide tighter bounds for pruning. In particular, some rely on a lower bound of the size of maximum DPlex and prefer a tight one. We remark that `DPHeuris` can provide a tight lower bound in polynomial time and thus significantly boosts the performance of `DPBB`. Our main contributions are as follows.

- We define a new model called directed (k, ℓ)-plex, and firstly study the problem of finding the maximum DPlex. We formally prove its NP-hardness (Section II).
- We design an efficient heuristic method, called `DPHeuris`, which finds a near-maximum DPlex. `DPHeuris` achieves a polynomial time complexity of $O(m \cdot n^2 \log n)$, where m and n are number of edges and vertices in the given directed graph, respectively (Section III).
- We further develop an efficient branch-and-bound algorithm, called `DPBB`, which finds the exact maximum DPlex. To boost its performance, we propose some effective techniques including 1) tailored graph reduction strategies and 2) new upper bounding methods (Section IV).
- We conduct extensive experiments on real directed graphs to verify the efficiency and scalability of our proposed algorithms `DPHeuris` and `DPBB` (Section V).

Among other sections, Section VI reviews the related work and Section VII concludes the paper.

II. Preliminaries

Let $G = (V, E)$ be a *directed* graph with the vertex set V and the edge set E where $|V| = n$ and $|E| = m$. We focus on the simple graphs, i.e., there is no self-loops and parallel edges. Given a directed edge $(u, v) \in E$, the direction is from vertex u to vertex v, i.e., $u \rightarrow v$; vertex u is an *in-neighbor* of vertex v, and correspondingly vertex v is an *out-neighbor* of vertex u. Given $v \in V$, the sets of *in-neighbors* and *out-neighbors* of v are denoted by $N_G^{in}(v)$ and $N_G^{out}(v)$, respectively; the *in-degree* and *out-degree* of a vertex v in G are denoted by $d_G^{in}(v) = |N_G^{in}(v)|$ and $d_G^{out}(v) = |N_G^{out}(v)|$, respectively. Given $S \subseteq V$, we use $G[S]$ to denote the subgraph induced by S, i.e., $G[S]$ includes the set of vertices S and the set of edges $\{(u, v) \mid (u, v) \in E \text{ and } u, v \in S\}$. All subgraphs considered in this paper are induced subgraphs. Given a subgraph H of G, we use $V(H)$ and $E(H)$ to denote the set of vertices and edges in H, respectively.

In this paper, we focus on the following cohesive subgraph:

Definition 1 (Directed (k, ℓ)-Plex (DPlex))**.** *An induced subgraph H is said to be a* directed (k, ℓ)-plex (DPlex) *for any two positive integers k and ℓ if*

- $d_H^{out}(v) \geq |V(H)| - k$ *for all* $v \in V(H)$, *and*
- $d_H^{in}(v) \geq |V(H)| - \ell$ *for all* $v \in V(H)$.

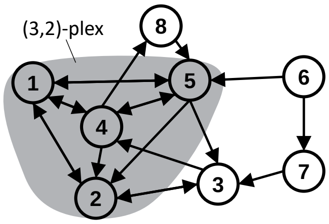


Fig. 1. An example graph

The intuition behind the cohesive pattern of directed (k, ℓ)-plex is that each vertex is allowed to disconnect at most a small number of vertices wrt both in-neighbors and out-neighbors. The values of k and ℓ are used to control the required cohesiveness of the target subgraph. It is easy to see that the smaller values of k and ℓ the denser subgraph.

We define a notion of *pseudo-degree* denoted by $pd_G(v)$ for each vertex v in graph G as follows.

$$pd_G(v) = \min(d_G^{in}(v) + \ell, d_G^{out}(v) + k)$$

It is easy to see that a directed subgraph $H = G[S]$ is a DPlex if and only if $pd_H[v] \geq |S|$ for all $v \in S$. Besides, we omit the subscript G from the notations when the context is clear.

Consider the example in Figure 1. The subgraph induced by $\{v_1, v_2, v_4, v_5\}$ is a directed $(3, 2)$-plex of size 4 (but not a directed $(2, 3)$-plex) where the pseudo-degree of v_2 is 4 for a directed $(3, 2)$-plex (3 for a directed $(2, 3)$-plex).

Hereditary Property. We note that DPlex satisfies the *hereditary property*, which will be utilized in our algorithmic design.

Lemma 1. *If G is a directed (k, ℓ)-plex, any subgraphs $H \subseteq G$ is a directed (k, ℓ)-plex.*

The above lemma can be easily obtained by the fact that each remaining vertex has the numbers of disconnected in-neighbors and out-neighbors non-increasing after excluding a set of vertices from a directed (k, ℓ)-plex.

We are ready to define the problem we study in this paper.

Problem Statement. Given a directed graph $G = (V, E)$ and two positive integers k and ℓ, the *Maximum Directed (k, ℓ)-Plex Search Problem (MDPS)* aims to find the largest directed (k, ℓ)-plex in G.

Another closely related concept is *maximality*. A directed (k, ℓ)-plex $G[S] \subseteq G[V]$ is said to be *maximal* if and only if there is no subgraph $G[S']$ such that 1) $S \subset S' \subseteq V$, and 2) $G[S']$ is also a directed (k, ℓ)-plex. It is easy to see that a maximum directed (k, ℓ)-plex is maximal while a maximal directed (k, ℓ)-plex is not necessarily maximum.

Hardness. We then show the NP-hardness of the problem.

Theorem 1. *The MDPS Problem is NP-hard.*

Proof. See Appendix of the technical report [20]. □

III. Heuristic Solutions

In this part, we introduce our new heuristic method, called `DPHeuris`. We first show the framework of `DPHeuris` in Section III.A It consists of (1) the graph reduction techniques

Algorithm 1: `DPHeuris`(G, k, ℓ)

```
   Input: a directed graph $G$, and two integers $k, \ell$
   Output: the size of a directed $(k, \ell)$-plex
1  $lb \leftarrow$ FastHeuris$(G, k, \ell)$;
2  GraphReduction$(G, k, \ell, lb)$;
3  do
4      $newlb \leftarrow$ StrongHeuris$(G, k, \ell, lb)$;
5      if $newlb > lb$ then
6          $lb \leftarrow newlb$;
7          GraphReduction$(G, k, \ell, lb)$;
8  while $lb$ is updated due to Line 6;
9  return $lb$;
```

and (2) lower bound computation methods for DPlex where the lower bound is the size of of a (not necessarily maximum) DPlex. We then present our two proposed lower bound computation methods, namely `FastHeuris` and `StrongHeuris`, in Section III.B. In particular, `FastHeuris` can quickly produce a lower bound while `StrongHeuris` may generate a better lower bound with a higher time cost.

A. *The Framework of* `DPHeuris`

Our framework of `DPHeuris` is shown in Algorithm 1. The high-level idea of our proposed `DPHeuris` is an iteraitve framework: In each iteration, we make use of lower bound computation methods to obtain a better lower bound lb and utilize the graph reduction techniques to remove vertices and edges that are unpromising to appear in a directed (k, ℓ)-plex whose size is larger than lb. In particular, for the lower bound computation methods, we call `FastHeuris` in Line 1 and iteratively call `StrongHeuris` in Line 4. Whenever we obtain a larger lower bound lb, we reduce the current graph using graph reduction techniques with lb in Lines 2 and 7.

Remark 1. We use two different lower bound computation methods in `DPHeuris`. This is because, as the size of the initial graph is relative large, we need a fast method, i.e., `FsatHeuris`, with $O(m + n)$ time to quickly find a lower bound, which can be used to delete unpromising vertices and edges. Then, based on the reduced graph with relative small size, our strong method `StrongHeuris` with $O(mn \log n)$ time can be iteratively utilized to obtain a better lower bound.

Remark 2. Note that our `DPHeuris` in Algorithm 1 returns the lower bound (Line 9) which corresponds to the size of a (not necessarily maximum) DPlex. However, it is straightforward to modify our algorithm in order to return the corresponding DPlex (with size lb).

Graph Reduction Methods. Graph reduction is a common practice when solving the maximum subgraph search problems. Specifically, based on a lower bound lb, it will reduce the input graph by removing those vertices/edges that cannot be in any optimum solution. Below, we adapt existing graph reduction techniques proposed for undirected graphs [19], [27] to solve our problem.

Lemma 2. *Given a directed graph $G = (V, E)$ and a lower bound lb, a vertex v cannot be included in a directed (k, ℓ)-plex of size greater than lb if $pd_G(v) \leq lb$.*

Proof. See Appendix of the technical report [20]. □

To facilitate the following lemma, we denote $\Delta_G^{in}(u, v)$ (resp., $\Delta_G^{out}(u, v)$) as the number of the common in-neighbors (resp., out-neighbors) of u and v in G, namely $|N_G^{in}(u) \cap N_G^{in}(v)|$ (resp., $|N_G^{out}(u) \cap N_G^{out}(v)|$).

Lemma 3. *Given a directed graph $G = (V, E)$ and a lower bound lb, we consider the following two cases according to the relationship between a pair of vertices u and v:*

- *Both edges (u, v) and (v, u) exist in E: if $\Delta_G^{in}(u, v) \leq lb - 2\ell$ or $\Delta_G^{out}(u, v) \leq lb - 2k$, then vertices u and v cannot be included in a directed (k, ℓ)-plex of size greater than lb at the same time.*
- *Exactly one of edges (u, v) and (v, u) exists in E: if $\Delta_G^{in}(u, v) \leq lb - 2\ell + 1$ or $\Delta_G^{out}(u, v) \leq lb - 2k + 1$, then vertices u and v cannot be included in a directed (k, ℓ)-plex of size greater than lb at the same time.*

Proof. See Appendix of the technical report [20]. □

We observe that the performance of the above graph reduction techniques depends on the lower bound lb. However, existing studies [19], [27] devote little effort to finding a tight lower bound. We remark that our proposed heuristic strategies (in Section III.B), namely `FastHeuris` and `StrongHeuris`, can find a tight lower bound and thus will further boost the performance of Lemma 2 and Lemma 3, as verified by our experiments in Figure 13(b) and Table III.

Time Complexity of `DPHeuris`. The number of iterations of Lines 3-8 is bounded by $O(n)$ as $lb = O(n)$. Each iteration calls `StrongHeuris` (Line 4) and `GraphReduction` (Line 7) once. Specifically, the procedure `GraphReduction` implements Lemma 2 and Lemma 3 in $O(m^{1.5})$. In particular, the time is dominated by 1) listing all triangles in the graph ([29], [35]) to compute $\Delta_G^{out}(\cdot, \cdot)$ and $\Delta_G^{in}(\cdot, \cdot)$; 2) update $\Delta_G^{out}(\cdot, \cdot)$ and $\Delta_G^{in}(\cdot, \cdot)$ when removing an edge or a vertex. To sum up, the time complexities of `FastHeuris`, `StrongHeuris` (the detailed analyses are deferred to Section III.B) and `GraphReduction` are $O(m + n)$, $O(mn \log n)$ and $O(m^{1.5})$ respectively. Due to $m = O(n^2)$, the time complexity of `DPHeuris` is thus $O(n \cdot (mn \log n + m^{1.5})) = O(mn^2 \log n)$.

B. *Lower Bound Computations*

We propose two methods for computing a lower bound of the maximum DPlex by returning the size of a (not necessarily maximum) DPlex. The first method is a fast heuristic with linear time complexity of $O(m+n)$ while the second heuristic method normally produces a DPlex with a larger lower bound using a higher time consumption of $O(mn \log n)$.

A Fast Heuristic Method. Our first method is a fast heuristic which is based on a simple observation from our definition of

```
Algorithm 2: FastHeuris($G = (V, E), k, \ell$)
   Input: a directed graph $G$, and two integers $k, \ell$
   Output: a lower bound of the maximum directed
           $(k, \ell)$-plex
1  Initialize $S \leftarrow V$;
2  while $G[S]$ is not a directed $(k, \ell)$-plex do
3      $u \leftarrow \arg\min_{v \in S} pd_{G[S]}(v)$;
4      $S \leftarrow S \setminus \{u\}$;
5      foreach $v \in N^{out}_{G[S]}(u) \cup N^{in}_{G[S]}(u)$ do
6          Update $pd_{G[S]}(v)$;
7  return $|S|$;
```

pseudo-degree: A directed subgraph $H = G[S]$ is a directed (k, ℓ)-plex if and only if we have $pd_H(v) \geq |S|$ for all $v \in S$.

Our fast heuristic is shown in Algorithm 2. We iteratively remove the vertex with the smallest pseudo-degree in the current graph (in Lines 3-6) until the remaining graph forms a DPlex. Then we return the size of such a directed (k, ℓ)-plex, which serves a lower bound of the maximum DPlex.

Analysis. The correctness of Algorithm 2 is easy to verify. We then analyze the time complexity of Algorithm 2. In a straightforward implementation, Algorithm 2 requires $O(m \log n)$ time as the number of pseudo-degree updates (Lines 5-6) is bounded by m and each update requires $O(\log n)$ time if a self-balancing binary search tree is used to store the pseudo-degree of each vertex. However, a crucial observation is that the pseudo-degree of a vertex in each update (Lines 5-6) is decreased by at most 1. Thus Algorithm 2 can be implemented in linear time of $O(m + n)$ with liner space of $O(m + n)$ by using a similar way of the sorted degree array in [4], [44].

A Strong Heuristic Method. FastHeuris in Algorithm 2 can compute a lower bound of the maximum directed (k, ℓ)-plex in linear time. However, in some cases, FastHeuris is not effective to compute a larger lower bound due to the following reasons:

- An interesting observation is that the directed (k, ℓ)-plex $G[S]$ computed in Algorithm 2 is not necessarily maximal. In other words, we may include one or more vertices into this obtained DPlex to get a larger DPlex.
- Algorithm 2 iteratively removes vertices in Lines 3-4. The starting point in this deletion-based strategy (i.e, the first vertex to be removed) is the vertex with the smallest pseudo-degree. However, the first removed vertex may also be in a larger DPlex. This means that different starting points affects the obtained DPlexes in Algorithm 2.
- In each iteration of Algorithm 2 (Lines 2-6), the selection criteria for removing a vertex is solely based on the pseudo-degrees for all vertices which overlooks the connection relations in the obtained DPlex.

To improve FastHeuris, we present our second heuristic method StrongHeuris. The high-level idea of StrongHeuris is to compute a lower bound of the maximum DPlex by considering maximal DPlexes that are extended from different starting vertices. The extension strategy is based on both connection relations in the current extending DPlex and pseudo-degrees in the graph.

```
Algorithm 3: StrongHeuris($G = (V, E), k, \ell, lb$)
   Input: a directed graph $G$, two integers $k, \ell$, and a
          given lower bound $lb$
   Output: a lower bound of the maximum directed
           $(k, \ell)$-plex
1  Sort vertices in non-increasing order of pseudo-degree.
     W.l.o.g., we assume $pd(v_1) \geq pd(v_2) \geq \cdots \geq pd(v_{|V|})$;
2  for $i \leftarrow 1$ to $|V|$ do
3      $size \leftarrow$ Extend$(G, v_i)$;
4      if $size > lb$ then
5          return $size$;
6  return $lb$;

   procedure: Extend($G = (V, E), w$)
   Output: the size of a maximal directed $(k, \ell)$-plex
           containing vertex $w$
   /* $S$: the vertex set in an extending
      DPlex; $P$: the candidate vertex set.  */
7  Initialize $S \leftarrow w$; $P \leftarrow V \setminus S$;
8  Let $nc(v)$ be the sum of in- and out-degrees of vertex
     $v$ in $G[S \cup \{v\}]$, i.e.,
     $nc(v) = d^{in}_{G[S \cup \{v\}]}(v) + d^{out}_{G[S \cup \{v\}]}(v)$ for each $v \in V$;
9  while $P \neq \emptyset$ do
10     Select vertex $u$ with largest pseudo-degree from
         the vertices that have largest value of $nc(\cdot)$ in $P$;
11     if $G[S \cup \{u\}]$ is a directed $(k, \ell)$-plex then
12         $S \leftarrow S \cup \{u\}$; Update $nc(\cdot)$;
13     $P \leftarrow P \setminus \{u\}$ ;
14 return $|S|$;
```

In particular, StrongHeuris is shown in Algorithm 3. We first sort vertices in non-increasing order of pseudo-degree in Line 1. Then, Algorithm 3 in Lines 2-5 enumerates starting vertices for the DPlex extension procedure Extend. In the i-th iteration, Extend with v_i is called and returns the size of a maximal DPlex containing v_i in Line 3. The algorithm returns the size obtained in Extend immediately once the size is larger than the input lower bound lb (Lines 4-5). Finally, Algorithm 3 returns lb in Line 6 if we cannot obtain a maximal DPlex with size larger than lb. Remark that the goal of StrongHeuris is to obtain a larger lower bound than the input lb. Specifically, we return immediately once we find such a better lower bound in Line 5. This is because the above introduced graph reduction techniques can remove several useless vertices and edges even if the lower bound is increased by only 1. Moreover, the vertices are sorted by pseudo-degree in non-increasing order in Line 1 since a vertex with larger pseudo-degree is more likely to be extended to a larger DPlex which implies that StrongHeuris may return a better lower bound earlier.

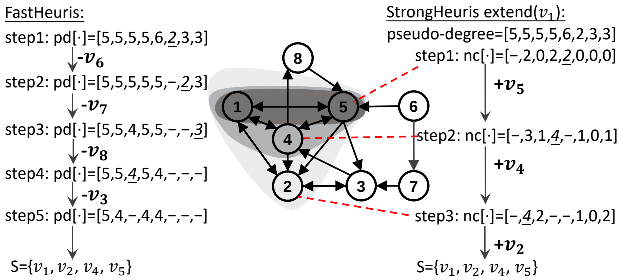

FastHeuris:
step1: pd[·]=[5,5,5,5,6,2,3,3]
-v_6
step2: pd[·]=[5,5,5,5,5,–,2,3]
-v_7
step3: pd[·]=[5,5,4,5,5,–,–,3]
-v_8
step4: pd[·]=[5,5,4,5,4,–,–,–]
-v_3
step5: pd[·]=[5,4,–,4,4,–,–,–]
S={v_1, v_2, v_4, v_5}
8
1
5
6
4
2
3
7
StrongHeuris extend(v_1):
pseudo-degree=[5,5,5,5,6,2,3,3]
step1: nc[·]=[–,2,0,2,2,0,0,0]
+v_5
step2: nc[·]=[–,3,1,4,–,1,0,1]
+v_4
step3: nc[·]=[–,4,2,–,–,1,0,2]
+v_2
S={v_1, v_2, v_4, v_5}

Fig. 2. An example for DPHeuris when (k, ℓ)=(3,2)

The Extend Procedure. Extend (shown in Lines 7-14 of Algorithm 3) first initializes $S \leftarrow w$ as the vertex set in an extending DPlex and P as the candidate vertex set. Then Extend iteratively includes a vertex u into S if $G[S \cup \{u\}]$ is a DPlex in Lines 9-13. Different strategies for the selection of the vertex u in each iteration (Line 10) produce different results. In particular, our vertex selection strategy is based on the following criteria.

- Vertex $u \in P$ has the largest number of neighbors in S (including both in- and out-neighbors), i.e., u belongs to the set of A=$\{v \in P \mid nc(u) \leq nc(v), \forall u \in P\}$ where $nc(u)$ is the count of neighbors of u in S.
- Based on the above criterion, vertex u has the largest value of pseudo-degree, i.e., $u = \arg\max_{v \in A} pd_G(v)$.

The reasons of our vertex selection strategy are as follows. The first criterion specifically considers the number of neighbors in S for each vertex in the candidate set P. Clearly, a vertex with larger number of neighbors will potentially be extended to a larger DPlex. Notice that this criterion focuses on the connection relations between a vertex in P and the current DPlex. For the second criterion, a vertex with larger value of pseudo-degree in G is more likely be in a larger DPlex.

Analysis. The correctness of Algorithm 3 follows from the facts that (1) Extend always returns the size of a maximal DPlex containing a specific vertex, and (2) the size of a maximal DPlex is always a lower bound of the optimum size of the MDPS problem. For the time complexity, we remark that Extend is called at most $|V| = n$ times in Algorithm 3. For each call of Extend, there are at most n iterations in Lines 9-13. The vertex selection in Line 10 can be implemented using $O(\log n)$ time using a standard self-balancing binary search tree. For the update of the array $nc(\cdot)$, we know the number of updates is at most m in total and each update requires $O(\log n)$ time since we use a standard self-balancing binary search tree. Thus, each Extend requires $O(n \log n + m \log n) = O(m \log n)$ time. Moreover, Line 1 of Algorithm 3 costs $O(n)$ time to sort the vertices (since the value of pseudo-degree is bounded by $O(n)$). Therefore, the time complexity of Algorithm 3 is thus $O(n + n \cdot m \log n) = O(mn \log n)$.

An Example for Heuristic Methods. Consider the example in Figure 2 with $k = 3$ and $\ell = 2$ using FastHeuris (Algorithm 2) and StrongHeuris (Algorithm 3).

We first consider FastHeuris where the array for storing pseudo-degrees of each vertex is $[5, 5, 5, 5, 6, 2, 3, 3]$. According to (Line 3 of) Algorithm 2, we first remove vertex v_6 as it has the smallest pseudo-degree of 2. Now the updated pseudo-degree array is $[5, 5, 5, 5, 5, -, 2, 3]$. FastHeuris repeats this process until we get the DPlex with $\{v_1, v_2, v_4, v_5\}$.

We next consider StrongHeuris with $lb = 4$. We consider to use v_1 in Line 3 of Algorithm 3 to extend to a maximal DPlex using Extend (Lines 7-14 of Algorithm 3). In Line 10 of Algorithm 3, the array $nc[\cdot]$ is $[-, 2, 0, 2, 2, 0, 0, 0]$ and the array $pd_G[\cdot]$ is $[5, 5, 5, 5, 6, 2, 3, 3]$. According to the vertex selection strategy in Line 10 of Algorithm 3, we first select the vertices v with greatest $nc[v]$, namely $\{v_2, v_4, v_5\}$, then among those vertices we choose the vertex with greatest pseudo-degree, i.e., we choose and include v_5 into the extending vertex set S as $\{v_1, v_5\}$ corresponds to a DPlex. Extend iteratively implements the above operations and the order of vertices that we try to include into S in Line 10 is $\{v_5, v_4, v_2, v_3, v_8, v_6, v_7\}$. Finally, we get S of $\{v_1, v_2, v_4, v_5\}$ which corresponds to a maximal DPlex containing v_1 in G. After performing all the vertices in Lines 2-5 of Algorithm 3, StrongHeuris finds there is no larger DPlexes obtained by Extend and returns $lb = 4$ in Line 6.

IV. A Branch-and-Bound based Exact Solution

We design a branch-and-bound method, called DPBB, for finding the maximum DPlex exactly. In particular, we introduce the framework of DPBB in Section IV.A and then present some techniques including effective graph reduction strategies for boosting the performance of DPBB in Section IV.B.

A. The Framework of DPBB

DPBB is shown in Algorithm 4, which runs in three consecutive stages, namely, Stage-I: Preprocessing, Stage-II: Divide-and-Conquer (DC) decomposition, and Stage-III: Branch-and-Bound search. Specifically, in Stage-I of preprocessing (Line 1), the input graph is reduced to a smaller one by invoking GraphReduction(G, lb) with the parameter of lower bound lb obtained by DPHeuris (note that DPHeuris finds a tight lower bound in practice and thus boosts the performance of GraphReduction, i.e., pruning more vertices). In Stage-II of DC decomposition (Lines 3-5), it further divides the reduced graph into multiple smaller ones. Then, in Stage-III of the branch-and-bound search (Lines 7-22), it runs a branch-and-bound solver called BnB on each of the small graphs to find the maximum DPlex exactly. In particular, we represent a branch in BnB by a pair of vertex sets (S, C), where S is the extending vertex set that represents the current DPlex and the candidate vertex set C includes vertices that are used to potentially extend the current DPlex. Below, we elaborate on the details of Stage-II and Stage-III.

Stage-II: The DC Decomposition. The DC decomposition (Lines 3-5 of Algorithm 4) is widely used for cohesive subgraph search problems (e.g., [11], [13], [49], [50]). Specifically, it divides the problem of finding the maximum DPlex on the input graph G to n sub-problems each on a smaller subgraph. The i-th ($1 \leq i \leq n$) sub-problem is on a smaller subgraph $G[\{v_i, v_{i+1}, ..., v_n\}]$ and aims to find a maximum DPlex that includes the vertex $\{v_i\}$ and does not include the vertices $\{v_1, ..., v_{i-1}\}$. We note that the maximum DPlex in

```
Algorithm 4: DPBB($G = (V, E), k, \ell$)
   Input: a directed graph $G$, two integers $k, \ell$, and a
          given lower bound $lb$
   Output: the size of the maximum directed $(k, \ell)$-plex
   // the preprocessing stage
1  $lb \leftarrow$ DPHeuris($G, k, \ell$);GraphReduction($G, lb$);
   // all recursive procedures share one
      global $G$
2  while $|V(G)| > lb$ do
3      $v_i \leftarrow \arg\min_{v \in V(G)} pd_G(v)$;
4      $lb \leftarrow \max\big(lb,$ BnB$(G, \{v_i\}, V(G) \setminus \{v_i\}, lb)\big)$;
5      Remove $v_i$ from $G$;
6  return $lb$;

   procedure: BnB($G, S, C, lb$)
   Output: the larger of $lb$ and the size of the maximum
           directed $(k, \ell)$-plex containing $S$ in $G[S \cup C]$
   /* $S$: the vertex set in an extending
      DPlex; $C$: the candidate vertex set.  */
7  Reduce $C$ using Reduction Rules;
8  $ub \leftarrow$ EstimateUB($G, S, C$);
9  if $ub \leq lb$ then return $lb$;
10 $u \leftarrow \arg\min_{v \in C} pd_{G[S \cup C]}(v)$;
11 $size \leftarrow$BnB($G, S, C \setminus \{u\}, lb$);
   // if $size > lb$, then $G$ was reduced by RAU
12 if $size > lb$ then
13     $lb \leftarrow size$;
14     if $S \nsubseteq V(G)$ then return $lb$;
15     $C \leftarrow C \cap V(G)$;
16 if $G[S \cup \{u\}]$ is not a DPlex then return $lb$;
17 if $|S \cup \{u\}| > lb$ then
18     $lb \leftarrow |S \cup \{u\}|$;
19     GraphReduction($G, k, \ell, lb$);
20     $C \leftarrow C \cap V(G)$;
21 $size \leftarrow$ BnB$(G, S \cup \{u\}, C \setminus \{u\}, lb)\big)$;
22 return $\max(lb, size)$;
```

the input graph is the largest one among all DPlexes found by the formed sub-problems. This is because 1) let i' be the smallest index such that vertex $v_{i'}$ is in the maximum DPlex in the input graph; 2) the maximum DPlex in the input graph then can be obtained via finding the maximum DPlex in the i'-th sub-problem with the subgraph $G[\{v_{i'}, v_{i'+1}, ..., v_n\}]$. Then each sub-problem is solved by invoking BnB with $S = \{v_i\}$ and $C = V(G) \setminus \{v_1, ..., v_{i-1}, v_i\}$ (Line 4). Besides, the performance of DC decomposition relies on the ordering of vertices $\{v_1, v_2, ..., v_n\}$. To boost it, we form the i-th sub-problem by choosing the vertex v_i that has the smallest pseudo-degree in the current graph (Line 3), as inspired by existing studies [11], [13], [49], [50].

Stage-III: The Branch-and-Bound search BnB.

BnB is essentially based on the depth-first search tree, and equipped with branching, reduction and bounding methods that are designed for directed graphs. Further, all recursive procedures of BnB share one global G (which is utilized in our newly proposed *RAU (Reduction After Updating lb) strategy*).

BnB recursively selects a *branching* vertex u (Line 10) and creates two branches (Lines 11 and 21): 1) excluding u from the current graph $G[S \cup C]$, and 2) including u into the extending vertex set S. It is then naturally to capture the relationship among branches in the depth-first search tree as the parent branch invokes the child branch using a branching vertex. At each branch of BnB, we apply *reduction rules* (Line 7) to remove unpromising vertices from the candidate set C, and estimate the *upper bound* ub of the maximum DPlex containing S (Line 8). It is clear that if $ub \leq lb$, we can prune the current branch and directly return with lb since the current branch with S cannot find a DPlex of size larger than lb (Line 9). Furthermore, if the current extending vertex set S with the branching vertex u cannot form a DPlex, we can return the current lower bound in Line 16 directly (due to the hereditary property of DPlex), i.e., BnB will not enter the branch that includes u. The branching, reduction and bounding methods that are used in DPBB will be discussed in Section IV.B.

Moreover, our newly proposed *RAU strategy* is implemented in Lines 12-15 and Lines 17-20 of DPBB, which aims to reduce the number of calls of GraphReduction during recursive calls to BnB. We introduce the RAU strategy as follows.

The RAU (Reduction After Updating lb) Strategy. The RAU strategy is designed based on the following observations. First, as the lower bound lb increases, there are more unpromising vertices and edges that can be removed using Lemmas 2 and 3 by invoking GraphReduction. Second, the time complexity of GraphReduction is $O(m^{1.5})$, which is rather expensive especially when we need to invoke at each branch. In addition, for the traditional branch-and-bound algorithm, if a child branch is applied reduction rules to remove vertices/edges, the effect of this reduction cannot be passed to the parent branch and the ancestor branches when the child branch returns. To solve above issues, we design the RAU strategy in order to invoke GraphReduction at a proper time and make the effect of reduction persistent for all the ancestor branches. The core idea of RAU is that 1) we only invoke GraphReduction when we find a larger lb, and 2) the graph G is global and is shared by all recursive BnB procedures such that the effect of reduction rules in the child branch can be passed to ancestor branches.

The RAU strategy is implemented in Algorithm 4. In RAU, GraphReduction is only called when a child branch gets a larger lb (Lines 17-20) and removes unpromising vertices/edges in G. Note that when this child branch returned, the parent branch would find that lb is changed and G is reduced. The ancestor branches find that lb is increased in Line 12, Algorithm 4 first checks whether $S \nsubseteq V(G)$ (Line 16); if it is the case, we can infer that the current extending set S has unpromising vertices. This means that S cannot be extended to a DPlex with size larger than lb which implies that the current branch can be safely pruned by directly returning

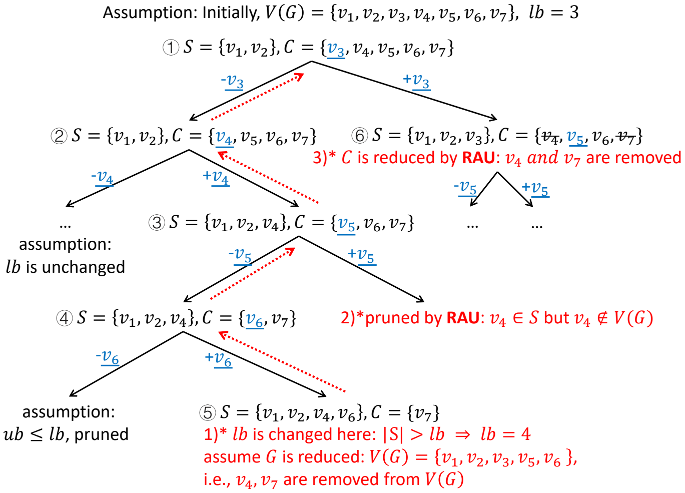

Assumption: Initially, $V(G) = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$, $lb = 3$
① $S = \{v_1, v_2\}, C = \{\underline{v_3}, v_4, v_5, v_6, v_7\}$
$\underline{-v_3}$
$\underline{+v_3}$
② $S = \{v_1, v_2\}, C = \{\underline{v_4}, v_5, v_6, v_7\}$
⑥ $S = \{v_1, v_2, v_3\}, C = \{\cancel{v_4}, \underline{v_5}, v_6, \cancel{v_7}\}$
3)* C is reduced by **RAU**: v_4 *and* v_7 are removed
$\underline{-v_4}$
$\underline{+v_4}$
$\underline{-v_5}$
$\underline{+v_5}$
...
assumption:
lb is unchanged
③ $S = \{v_1, v_2, v_4\}, C = \{\underline{v_5}, v_6, v_7\}$
...
...
$\underline{-v_5}$
$\underline{+v_5}$
④ $S = \{v_1, v_2, v_4\}, C = \{\underline{v_6}, v_7\}$
2)*pruned by **RAU**: $v_4 \in S$ but $v_4 \notin V(G)$
$\underline{-v_6}$
$\underline{+v_6}$
assumption:
$ub \leq lb$, pruned
⑤ $S = \{v_1, v_2, v_4, v_6\}, C = \{v_7\}$
1)* lb is changed here: $|S| > lb \Rightarrow lb = 4$
assume G is reduced: $V(G) = \{v_1, v_2, v_3, v_5, v_6\}$,
i.e., v_4, v_7 are removed from $V(G)$

Fig. 3. An example of BnB with the RAU strategy (the branching vertices are blue and underlined; the effects of RAU are red and marked with *; ①-⑥ represent the visiting order in the depth-first search tree of BnB; 1)-3) represent the execution order of RAU strategy)

lb in Line 14. Also, the candidate set C can be updated to remove vertices that are already removed from G in Line 15.

The correctness of the RAU strategy suffices to prove that any maximum DPlex will not be destroyed due to the RAU strategy. Note that in Line 19 of Algorithm 4, only vertices/edges that will not be in a DPlex with size larger than lb are removed from G by GraphReduction. Thus, if we find that $S \nsubseteq V(G)$ in Line 14, we can infer that $S \setminus V(G)$ are unpromising vertices that have been removed from G previously. In a similar way, since only unpromising vertices are removed from C in Lines 15 and 20, the reduction of C will not cause an incorrect solution.

An Example for RAU. Consider the example in Figure 3 where the root of the depth-first search subtree is associated with $S = \{v_1, v_2\}$, $C = \{v_3, v_4, v_5, v_6, v_7\}$ and $lb = 3$. Each time a branching vertex $u \in C$ (the underlined vertex) is selected to generate two branches: the left one excludes u from G and the right one includes u into S. For instance, the root selects v_3 and first enters the left branch which removes v_3 from G. Assume that we cannot find a larger DPlex until the branch associated with $S = \{v_1, v_2, v_4, v_6\}$ and $C = \{v_7\}$. We also assume that before we arrive at this branch, our pruning techniques may prune unpromising branches (i.e., the branches such that $ub \leq lb$). The red dotted arrows demonstrate the process of the RAU effect: 1) When we update lb to 4, suppose that v_4 and v_7 are removed from G due to RAU which means that v_4 and v_7 will not belong to any DPlex with size larger than lb; 2) the branch that includes v_4 can be pruned since v_4 is in S but not in G; 3) when we return to the root, v_4 and v_7 are removed from C since v_4 and v_7 are removed from G.

B. Branching, Reduction and Bounding

In this part, we discuss our branching, reduction and bounding strategies that are utilized in our proposed DPBB.

Branching Rule. We use the branching rule to select which vertex of C to be considered next in BnB. In particular, our branching rule generates two branches based on the selected vertex u: 1) the first branch excludes u from C, and 2) the second branch includes u into the current DPlex $G[S]$. The vertex u we select is the vertex with minimum pseudo-degree in C, and we first consider the branch which excludes u from C. The rationale of our branching rule is that we want to remove more unpromising vertices first such that we can quickly find a DPlex that has size larger than lb.

Reduction Rules. The reduction rules are used to remove unpromising vertices from the candidate set C in Line 7 of Algorithm 4. That is, given a DPlex $G[S]$ and a lower bound lb, a vertex u is unpromising if (1) $G[S \cup \{u\}]$ is not a DPlex, or (2) there is no DPlex that contains the vertex set $S \cup \{u\}$ and has size larger than lb.

We then give two types of reduction rules. The first type is the degree-based reduction rules which is based on Lemma 2:

Lemma 4. *Given a directed graph G and a directed (k, ℓ)-plex $G[S]$, a vertex $u \in C$ can be removed from G if u has at most $|S| - k$ out-neighbors or at most $|S| - \ell$ in-neighbors in the DPlex $G[S]$.*

Lemma 5. *Given a directed graph G, a directed (k, ℓ)-plex $G[S]$, and a lower bound lb, a vertex $u \in C$ can be removed from C if $pd_{G[S \cup C]}(u) \leq lb$.*

Proof of Lemma 4 and Lemma 5. See Appendix of the technical report [20]. □

In addition, our next reduction rule is based on the diameter of a DPlex as shown in the following lemma.

Lemma 6. *Let $G[S]$ be a directed (k, ℓ)-plex. If we have $|S| > k + \ell - 2$, then the diameter of $G[S]$ is no more than 2, i.e., $\forall u, v \in S$, the distance from u to v is at most 2.*

Proof. See Appendix of the technical report [20]. □

Note that the diameter-based reduction rule in Lemma 6 is applicable only when $|S| > k + \ell - 2$. Based on Lemma 6, we have the following reduction rule.

Lemma 7. *We are given a directed graph G, a directed (k, ℓ)-plex $G[S]$, a lower bound lb, and $lb > k + \ell - 3$. A vertex $u \in C$ can be removed from C if there exist $v \in S$ and $u \notin N^{\leq 2}_{G[S \cup C]}(v)$ where $N^{\leq 2}_{G[S \cup C]}(v)$ is the set of vertices in $G[S \cup C]$ that have distance at most 2 from the vertex v.*

Proof. See Appendix of the technical report [20]. □

Upper Bounds. Below, we will discuss several methods used for estimating the upper bound ub of the maximum DPlex containing the vertex set S. It is clear that once the estimated ub is no larger than the lower bound lb, we can safely prune the current branch that associates with the vertex set S.

We first adapt the existing upper bound technique proposed in [27] for the undirected graph to the directed graph, which is based on the following lemma:

Algorithm 5: `BinaryEstimate`$(G[S \cup \Pi_0])$

```
Input: the graph induced by the vertex set $S \cup \Pi_0$
Output: $ub$ : the upper bound of the maximum DPlex
        in $G[S \cup \Pi_0]$
1 Initialize $S' \leftarrow S \cup \Pi_0$; $L \leftarrow \min(k, \ell)$; $R \leftarrow |S'|$;
2 Compute $pd_{G[S']}(u)$ for each $u \in S'$ ;
3 while $L < R$ do
4     $mid \leftarrow (L + R + 1)/2$;
5     if $\Big|\{u|u \in S'$ and $pd_{G[S']}(u) \geq mid\}\Big| < mid$ then
6         $R \leftarrow mid - 1$;
7     else
8         $L \leftarrow mid$;
9 return $R$;
```

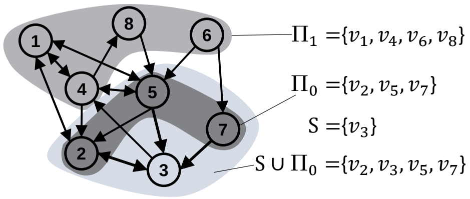

1
8
6
4
5
7
2
3
$\Pi_1 = \{v_1, v_4, v_6, v_8\}$
$\Pi_0 = \{v_2, v_5, v_7\}$
$S = \{v_3\}$
$S \cup \Pi_0 = \{v_2, v_3, v_5, v_7\}$

Fig. 4. An example for upper bound computations

Lemma 8. *Given a directed graph $G = (V, E)$ and a directed (k, ℓ)-plex $G[S]$ corresponding to the current branch (S, C), let $S = \{v_1, v_2, ..., v_{|S|}\}$. Based on the out-neighbors, $\Pi_0, \Pi_1, \Pi_2, ..., \Pi_{|S|}$ is a partition of C satisfying:*

- *$\Pi_i \cap \Pi_j = \varnothing$, when $i \neq j$;*
- *$\Pi_0 \cup \Pi_1 \cup \Pi_2 \cup ... \cup \Pi_{|S|} = C$;*
- *$(v_i, u) \notin E$, for $1 \leq i \leq |S|$ and $\forall u \in \Pi_i$;*
- *$(v_i, u) \in E$, $\forall u \in \Pi_0$ and $\forall v_i \in S$.*

The upper bound ub of the maximum directed (k, ℓ)-plex containing the vertex set S can be computed as: $ub = |S| + |\Pi_0| + \sum_{v_i \in S} \min(|\Pi_i|, k - |S \setminus N_G^{out}(v_i)|)$. Analogously, if C is partitioned into $\Pi_0, \Pi_1, \Pi_2, ..., \Pi_{|S|}$ based on the in-neighbors, we have $ub = |S| + |\Pi_0| + \sum_{v_i \in S} \min(|\Pi_i|, \ell - |S \setminus N_G^{in}(v_i)|)$.

Proof. See Appendix of the technical report [20]. □

We observe that the upper bound ub provided by Lemma 8 is not tight. The rationale is that: (1) ub is dominated by the size $|S| + |\Pi_0|$ in practice, (2) based on the proof of Lemma 8, the number of vertices in Π_0 that can be included into the current DPlex $G[S]$ is upper bounded by $|\Pi_0|$ which ignores the connection relations among vertices in $S \cup \Pi_0$.

The `BinaryEstimate` Algorithm. To tighten the upper bound, we further consider the connection relations among vertices in $S \cup \Pi_0$, and particularly focus on the maximum DPlex in $G[S \cup \Pi_0]$ (whose size is clearly an upper bound of the maximum DPlex containing S in $G[S \cup \Pi_0]$). Assume $G[S^*] \subseteq G[S \cup \Pi_0]$ is the maximum DPlex in $G[S \cup \Pi_0]$. We have that $pd_{G[S \cup \Pi_0]}(u) \geq pd_{G[S^*]}(u) \geq |S^*|$. In other words, there are at least $|S^*|$ vertices whose pseudo-degree is not less than $|S^*|$ in $G[S \cup \Pi_0]$. Based on this property, we use the binary search to get the upper bound of $|S^*|$ which is shown in Algorithm 5 (called `BinaryEstimate`). It is easy to verify the correctness of `BinaryEstimate` according to the above explanation. For the time complexity, Line 2 of Algorithm 5 requires $O(m + n)$ to compute the pseudo-degrees of all vertices. Then we can use an array of size $O(n)$ where the i-th entry stores the number of vertices that have pseudo-degrees at least i. The algorithm follows a binary search fashion in Lines 3-8 where each search needs $O(1)$ time to check based on the array we described. In total, the time complexity of Algorithm 5 is $O(m + n + \log n) = O(m + n)$.

Algorithm 6: `EstimateUB`(G, S, C)

```
Input: a directed graph $G$ and two vertex sets $S$ and $C$
Output: the upper bound $ub$ of the maximum DPlex
        containing $S$ in $G[S \cup C]$
1 $ub \leftarrow \min_{v \in S} pd_{G[S \cup C]}(v)$;
2 return $\min(ub,$ PartOut$(G, S, C)$, PartIn$(G, S, C))$;

  procedure: PartOut$(G, S, C)$
3 Initialize $P \leftarrow C$;
4 for $i = 1$ to $|S|$ do
5     $\Pi_i = \{u|u \in P$ and $(v_i, u) \notin E\}$;
6     $P \leftarrow P \setminus \Pi_i$;
7 $\Pi_0 \leftarrow P$;
8 $ub \leftarrow \sum_{v_i \in S} \min(|\Pi_i|, k - |S \setminus N_G^{out}(v_i)|) +$
    BinaryEstimate$(G[S \cup \Pi_0])$;
9 return $ub$;

  procedure: PartIn$(G, S, C)$
10 Initialize $P \leftarrow C$;
11 for $i = 1$ to $|S|$ do
12     $\Pi_i = \{u|u \in P$ and $(u, v_i) \notin E\}$;
13     $P \leftarrow P \setminus \Pi_i$;
14 $\Pi_0 \leftarrow P$;
15 $ub \leftarrow \sum_{v_i \in S} \min(|\Pi_i|, \ell - |S \setminus N_G^{in}(v_i)|) +$
    BinaryEstimate$(G[S \cup \Pi_0])$;
16 return $ub$;
```

Our Upper Bounding Method: `EstimateUB`. Finally, our proposed upper bound computation method, called `EstimateUB`, combines the ideas of Lemma 8 and `BinaryEstimate` (Algorithm 5), which is shown in Algorithm 6. `EstimateUB` computes the upper bound by considering the minimum pseudo-degree in $G[S \cup C]$ of vertex in S (Line 1) and the in-neighbor and out-neighbor partitions (Line 2) due to Lemma 8 and `BinaryEstimate`. The time complexity of Algorithm 6 is then $O(m + n + |S| \times |C|) = O(m + n^2)$ where the term of $O(|S| \times |C|)$ is due to the partition of the vertex set in Lines 3-7 and Lines 10-14.

An Example for Upper Bound Computations. Consider the running example in Figure 4 where (k, ℓ)=(3,2). We first consider the upper bound obtained using only Lemma 8. Suppose S=$\{v_3\}$. We make use of the in-neighbor partition, and get Π_0=$\{v_2, v_5, v_7\}$ and Π_1=$\{v_1, v_4, v_6, v_8\}$ where all vertices in Π_0 are in-neighbors of each vertex in S while

TABLE I
STATISTICS OF DATA SET

ID	Dataset	$\lvert V\rvert$	$\lvert E\rvert$	Density	d^{in}_{max}	d^{out}_{max}	$d^{out}_{avg}(d^{in}_{avg})$	$\lvert V(H_{4,4})\rvert$
G1	email-Eu-core	1,005	24,929	0.0247	211	333	24.8050	20
G2	Wiki-Vote	7,115	103,689	0.0020	457	893	14.5733	12
G3	p2p-Gnutella06	8,717	3,1525	0.0004	64	113	3.6165	5
G4	as-caida20040105	16,301	65,910	0.0002	2331	2331	4.0433	20
G5	enron	69,244	276,143	0.0001	1394	1392	3.9880	19
G6	soc-Epinions1	75,879	508,837	0.0001	3035	1801	6.7059	23
G7	Slashdot0811	77,360	828,161	0.0001	2539	2507	10.7053	37
G8	email-eu-all	264,142	417,867	0.0000	7597	929	1.5820	18
G9	Amazon0601	403,394	3,387,388	0.0000	2751	10	8.3972	12
G10	web-Google	875,713	5,105,039	0.0000	6326	456	5.8296	33
G11	imdb-2021	2,996,317	10,739,291	0.0000	833	833	3.5842	53
G12	soc-pokec-relationship	1,632,803	30,622,564	0.0000	13733	8763	18.7546	24
G13	sx-stackoverflow	2,584,164	34,875,684	0.0000	22257	38147	13.4959	44
G14	soc-LiveJournal1	4,847,571	68,475,391	0.0000	13905	20292	14.1257	254
G15	enwiki-2022	6,492,490	159,047,205	0.0000	231598	12010	24.4971	68
G16	it-2004	41,291,594	1,135,718,909	0.0000	1326744	9964	27.5048	3199

the vertices in Π_1 are not. In this case, we derive $ub = 5$ because Π_1 can provide at most ℓ-1=1 vertices. On the other hand, we can further utilize BinaryEstimate where the pseudo-degree array of $S \cup \Pi_0$=$\{v_2, v_3, v_5, v_7\}$ is $[4, 4, 2, 2]$. We then conclude that ub=1+2=3 where the term of 2 is due to BinaryEstimate. This example shows that using BinaryEstimate can potentially get a better upper bound.

V. EXPERIMENTAL STUDIES

Datasets. We use 16 real datasets which are taken from the website (https://law.di.unimi.it/datasets.php) and SNAP (http://snap.stanford.edu). We summarize the statistics of the datasets in Table I, where the density of a graph $G = (V, E)$ is defined by $|E|/(|V|{\cdot}(|V|{-}1))$, d^{in}_{max} and d^{out}_{max} (resp. d^{in}_{avg} and d^{out}_{avg}) denote the maximum (resp. the average) in-degree and out-degree, respectively, and $|V(H_{4,4})|$ denotes the number of vertices in the maximum (4,4)-plex within each dataset.

Algorithms. We evaluate the proposed DPHeuris and DPBB in terms of the total running times of algorithms and the sizes of returned DPlexes in Section III. Note that existing algorithms cannot be directly applied to our problem since either (1) they ignore the direction information of edges or (2) their designs only work for the particular directed models. Besides, in the ablation studies of Section III, we also evaluate some possible variants of DPHeuris and DPBB.

Setup. All algorithms are implemented in C++ and compiled with -O3 optimization. All experiments are run in the single-thread mode on a machine with an Intel(R) Xeon(R) Platinum 8358P CPU @ 2.60GHz and 256GB main memory running Ubuntu 20.04.6. We set the running time limit as 24 hours and use four datasets, namely, email-Eu-core, email-eu-all, soc-Epinions1, and imdb-2021, as default ones since they cover different graph scales. Besides, we set (k, ℓ) as $(4, 4)$ by default. Our code and datasets are available at https://github.com/ShuohaoGao/MDPS.

A. Case Study: Community Search

We investigate five types of directed cohesive subgraphs, namely DPlex, D-core [18], D-truss [32], directed clique (D-clique) [37] and directed (γ_1, γ_2)-quasi-clique (D-QC) [23], for a community search task. We use the communication network email-Eu-core (as shown in Table I), which is generated from email data of a research institution with 42 research departments (i.e., 42 ground-truth communities). Specifically, it first represents each member of the research institution by a vertex, and then adds a directed edge from member u to member v if member u sent at least one email to member v. Finally, email-Eu-core contains 24K email communications between 1K members. Given the graph and a query member v, the task aims to find the community (i.e., research department) that v belongs to. To solve the task, we find the maximum cohesive subgraph (e.g., maximum DPlex) containing v, and classify and return all members in the found subgraph as one community. For some types of cohesive subgraph, we explore different parameter settings (by varying k and ℓ). We evaluate the quality of the found community (i.e., subgraph $G[H]$) by measuring the precision, recall, F1-score, density, community size and community member similarities CMS^{in} and CMS^{out} defined as below.

$$CMS^{in}(H) = \frac{1}{|V(H)|^2} \sum_{u \in V(H)} \sum_{v \in V(H)} \frac{|N^{in}_H(u) \cap N^{in}_H(v)|}{|N^{in}_H(u) \cup N^{in}_H(v)|}$$

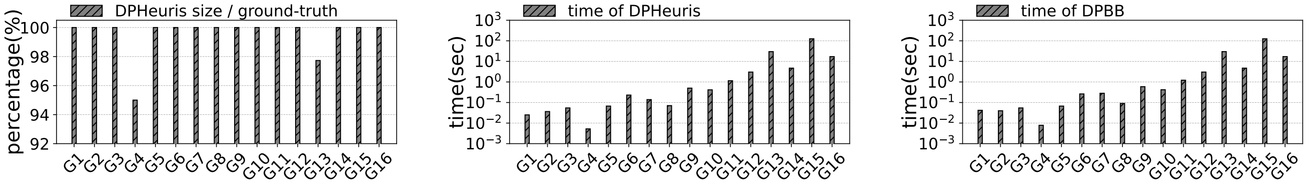
$$CMS^{out}(H) = \frac{1}{|V(H)|^2} \sum_{u \in V(H)} \sum_{v \in V(H)} \frac{|N^{out}_H(u) \cap N^{out}_H(v)|}{|N^{out}_H(u) \cup N^{out}_H(v)|}$$

We conduct 100 queries by selecting different query members, and report the average results in Table II. We have the following observations.

- **DPlex**. It achieves the best F1-score (0.596) when $(k, \ell) = (4, 4)$ among all methods. Besides, it has both high CMS^{in} and high CMS^{out} (e.g., around 0.72), which indicates the high similarity among all members in the found DPlex. This shows that a community can be effectively captured by finding the maximum DPlex. In addition, we find that DPlex has the precision decrease and the recall increase as (k, ℓ) increases. This is because DPlexes with a large (k, ℓ) would involve more disconnections and become less cohesive (see CMS) and larger (see size$_{avg}$).

TABLE II
QUALITY COMPARISON: DIFFERENT DIRECTED MODELS ON COMMUNITY SEARCH

model	precision	recall	F1-score	density	CMS^{in}	CMS^{out}	size$_{avg}$
directed clique	0.9384	0.3461	0.4947	**1.0000**	**0.7793**	**0.7793**	8.52
directed (2,2)-plex	0.9014	0.3891	0.5251	0.9238	0.7238	0.7237	9.37
directed (3,3)-plex	0.8893	0.4371	0.5692	0.9049	0.7297	0.7207	11.54
directed (4,4)-plex	0.8934	**0.4576**	**0.5960**	0.8896	0.7214	0.7150	13.12
directed (15,15)-core	0.0843	0.3909	0.1369	0.1041	0.0809	0.0698	402.00
directed (20,20)-core	0.0754	0.3545	0.1181	0.1902	0.1408	0.1279	236.00
directed (25,25)-core	0.0651	0.3200	0.0998	0.3688	0.2657	0.2519	118.00
directed (10,10)-truss	0.0877	0.3615	0.1352	0.2625	0.1930	0.1818	157.00
directed (11,11)-truss	0.0875	0.2785	0.1214	0.3802	0.2824	0.2674	104.00
directed (12,12)-truss	0.0765	0.2137	0.1013	0.3789	0.2810	0.2703	74.10
directed (0.85,0.85)-quasi-clique	0.9316	0.4031	0.5576	0.9682	0.7513	0.7515	9.78
directed (0.90,0.90)-quasi-clique	0.9419	0.3821	0.5331	0.9854	0.7611	0.7611	8.98
directed (0.95,0.95)-quasi-clique	**0.9456**	0.3550	0.5028	0.9987	0.7679	0.7679	8.54

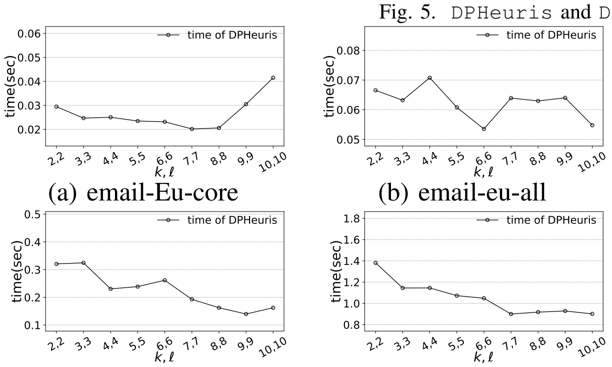


(a) Relative size of DPHeuris results

(b) Time of DPHeuris

(c) Time of DPBB

Fig. 5. DPHeuris and DPBB: All datasets when (k, ℓ)=(4,4)



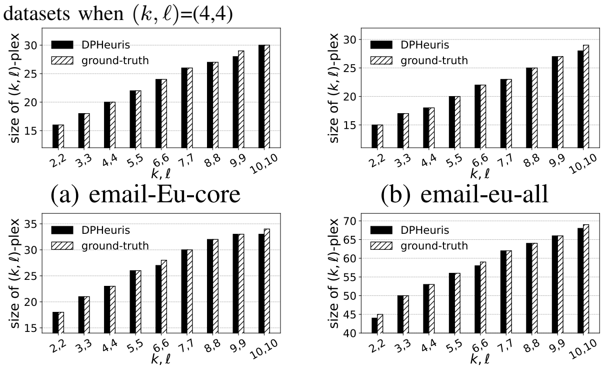
(a) email-Eu-core

(b) email-eu-all

(c) soc-Epinions1

(d) imdb-2021

Fig. 6. Time of DPHeuris: Varying k and ℓ



(a) email-Eu-core

(b) email-eu-all

(c) soc-Epinions1

(d) imdb-2021

Fig. 7. Size of DPHeuris: Varying k and ℓ

- **D-clique**. It has high precision but low recall (e.g., below 0.35), and has the F1-score of 0.497 lower than DPlex. This is because it requires full connections which is too strict.
- **D-core and D-truss**. They both have relatively high recall but very low precision (e.g., below 0.1 in all settings). This is because the found D-core and D-truss would be very large (see size$_{avg}$) and involve many disconnections (see density), thus being less cohesive (see CMS).
- **D-QC**. The precision decreases and the recall increases with smaller (γ_1, γ_2) since it would allow more disconnections in a D-QC. When $(\gamma_1, \gamma_2) = (0.85, 0.85)$, it achieves the best F1 score of 0.5576, which is worse than that of (4,4)-DPlex. Besides, we note that although D-QC achieves a comparable F1 score, finding D-QCs is harder and more time-consuming than finding DPlexes in both theory and practice [22], [50].

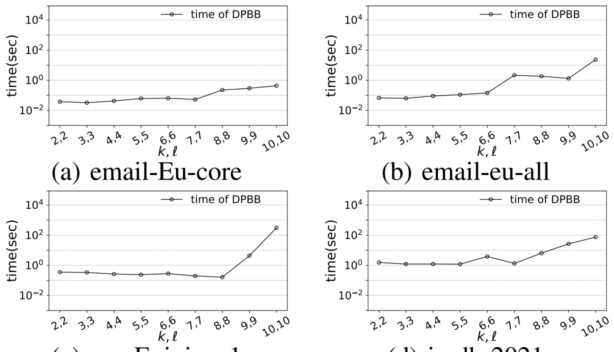
We also conduct another case study on citation prediction in the citation network, which is deferred to Appendix of the technical report [20] due to the space limit.

B. Efficiency Evaluation

DPHeuris: All datasets. We evaluate DPHeuris on all datasets, and report the size of returned DPlexes and the running time in Figure 5(a) and 5(b) . We have the following observations. First, it can handle all datasets within 200 seconds and has the running time increase as the scale of graph grows, which coincides with our time complexity analysis. This demonstrates the *efficiency* and *scalability* of DPHeuris. Second, it has the size of the found DPlexes close to that of the maximum ones (i.e., the ground-truth) on all datasets, which indicates the *effectiveness* of DPHeuris.

DPHeuris: Varying k and ℓ. We evaluate DPHeuris by varying the values of k and ℓ on default datasets, and report the running time and the size of returned DPlexes in Figure 6 and Figure 7, respectively. First, DPHeuris can handle all settings within 2 seconds. Specifically, the running times for different values of k and ℓ are similar. This is consistent with our theoretical analysis. Second, the sizes of the found DPlexes are closed to the ground-truth on all settings. In particular, the size of the found DPlexes and the ground-truth increase as the values of k and ℓ grows. This is because the larger k and ℓ are, the more disconnections inside a DPlex are allowed.

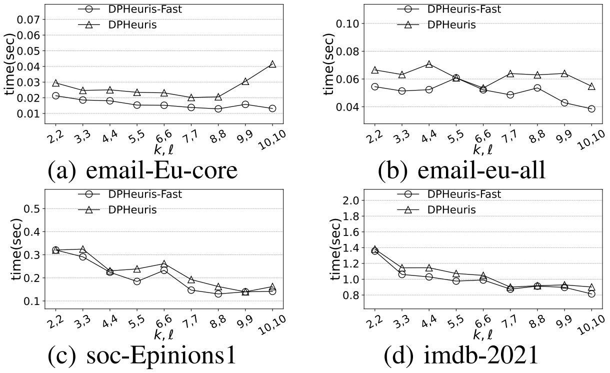
DPBB: All datasets. We evaluate the proposed exact algorithm DPBB on all datasets and show the running time in Figure 5(c). DPBB can find the maximum DPlex within 200 seconds on



(a) email-Eu-core (b) email-eu-all

(c) soc-Epinions1 (d) imdb-2021

Fig. 8. Time of DPBB: Varying k and ℓ



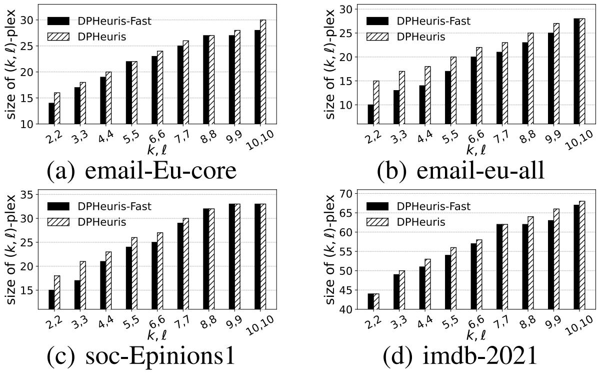
(a) email-Eu-core (b) email-eu-all

(c) soc-Epinions1 (d) imdb-2021

Fig. 9. Time of DPHeuris: Effect of iteration

all datasets. Besides, it runs slightly slower than the heuristic method DPHeuris. This is because DPBB (1) incorporates DPHeuris into a preproceesing procedure of reducing the scale of input graph and (2) achieves larger time complexity compare to DPHeuris in theory.

DPBB: Varying k and ℓ. We evaluate DPBB by varying the values of k and ℓ and report the time in Figure 8. We observe that DPBB is scalable to handle various settings of k and ℓ.



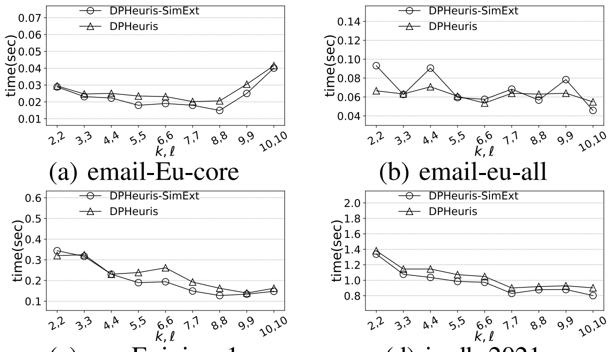
(a) email-Eu-core (b) email-eu-all

(c) soc-Epinions1 (d) imdb-2021

Fig. 10. Size of DPHeuris: Effect of iteration

C. Ablation Studies

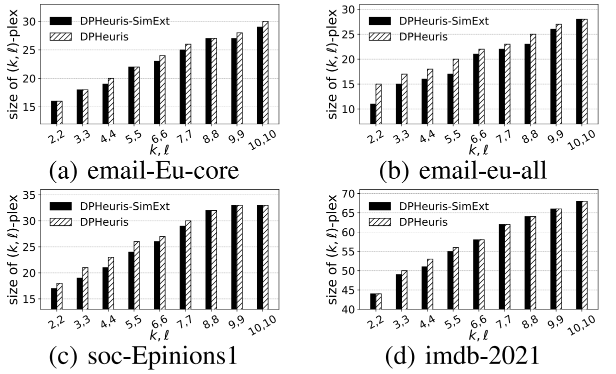
DPHeuris: Effect of iteration. We study the effect of the heuristic search process StrongHeuris on DPHeuris. To this end, we compare DPHeuris with its variant DPHeuris-Fast that only invokes FastHeuris and GraphReduction (Lines 1-2 in Algorithm 1). We report the running time and the size of found DPlexes in Figure 9 and Figure 10, respectively. We have the following observations. First, they run comparably fast, although DPHeuris runs slightly slower due to the multiple calls of StrongHeuris and GraphReduction. This is because the input graph would be reduced significantly by GraphReduction after FastHeuris and, therefore, the following rounds would run faster. Second, DPHeuris tends to find larger DPlexes compared to DPHeuris-Fast since it incorporates more computations into the heuristic search process.



(a) email-Eu-core (b) email-eu-all

(c) soc-Epinions1 (d) imdb-2021

Fig. 11. Time of DPHeuris: Comparison among various extending strategies



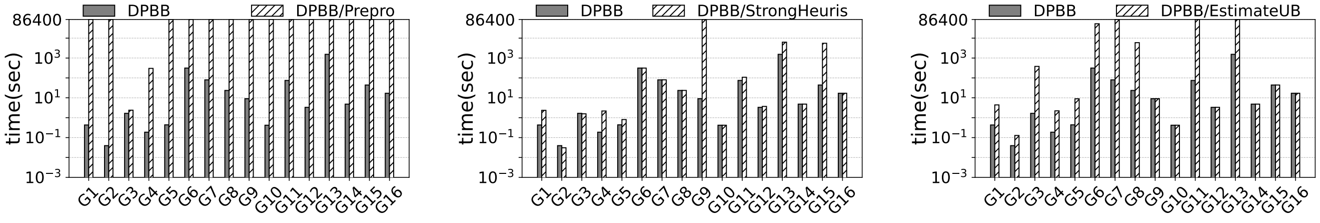
(a) email-Eu-core (b) email-eu-all

(c) soc-Epinions1 (d) imdb-2021

Fig. 12. Size of DPHeuris: Comparison among various extending strategies

DPHeuris: Comparison among various extending strategies. We study the effect of various extending strategies and compare DPHeuris with one variant DPHeuris-SimExt, i.e., DPHeuris with a simple extending strategy. Specifically, it simply changes the order of vertices that we try to include into the DPlex, and the order we used in DPHeuris-SimExt is the descending order sorted by pseudo-degree. The results are shown in Figure 11 for running time and in Figure 12 for the size of found DPlexes. We find that DPHeuris performs better. They run comparably fast but DPHeuris tends to find larger DPlexes. This is because our strategy considers more information (i.e., the neighbours) of each vertex for figuring out their priorities during the extending process.

DPBB: Effect of the preprocessing procedure. We study the effect of the preprocessing procedure and compare DPBB with one variant DPBB/Prepro, i.e., DPBB without the preprocessing procedure in Figure 13(a). DPBB performs better than DPBB/Prepro by achieving up to six orders of magnitude speedup. This is because the preprocessing procedure can significantly reduce the size of the input graph as shown in



(a) Effect of Preprocessing (b) Effect of StrongHeuris (c) Effect of EstimateUB

Fig. 13. Time of DPBB and its variants when (k, ℓ)=(10,10)

TABLE III
COMPARISON OF GRAPH SIZE WHEN (k, ℓ)=(4,4)

	Input		DPHeuris-Fast		DPHeuris	
ID	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert V_1\rvert$	$\lvert E_1\rvert$	$\lvert V_2\rvert$	$\lvert E_2\rvert$
G1	1.0K	25K	82	2661	66	1905
G2	7.1K	100K	45	894	26	414
G3	8.7K	32K	3226	13859	0	0
G4	16K	66K	25	504	25	504
G5	69K	270K	0	0	0	0
G6	76K	510K	188	8105	98	3716
G7	77K	830K	121	8911	109	7727
G8	260K	420K	173	6695	73	2071
G9	400K	3.4M	7350	64141	1942	18104
G10	880K	5.1M	64	1986	64	1986
G11	1.2M	11M	129	12008	113	10028
G12	1.6M	31M	29	740	29	740
G13	2.6M	35M	129	11742	129	11742
G14	4.8M	68M	262	68281	0	0
G15	6.5M	160M	11661	634600	0	0
G16	41M	1.1B	0	0	0	0

Table III and running DPBB on a smaller graph would be efficient. Note that for some graphs (e.g., G16), the numbers of vertices and edges are 0 after applying the preprocessing procedure (using either DPHeuris-Fast or DPHeuris) since it gets the optimum solution and the graph can then be reduced to be empty using reduction rules. This demonstrates the effectiveness of the preprocessing procedure.

DPBB: Effect of StrongHeuris. We compare the full version method DPBB with a variant DPBB/StrongHeuris (i.e., DPBB using DPHeuris-Fast which excludes the strategy StrongHeuris) in Figure 13(b). DPBB runs faster and achieves up to four orders of magnitude speedup on G9 with the setting $(k, \ell) = (10, 10)$. This is because DPHeuris with StrongHeuris can find a larger DPlex (see Figure 10), which boosts the performance of the pre-processing procedure and the graph reduction rules. This can also be verified by Table III that applying DPHeuris with StrongHeuris can result in a smaller graph than applying DPHeuris-Fast.

DPBB: Effect of EstimateUB. We study the effect of the proposed upper bounding techniques (Algorithm 6) and compare DPBB with one baseline DPBB/EstimateUB, i.e., DPBB without the upper bounding techniques in Figure 13(c). DPBB runs faster than DPBB/EstimateUB up to three orders of magnitude on G11, which indicates the effectiveness of the proposed upper bounding techniques.

VI. Related Works

The problems of cohesive subgraph search are extensively studied in the literature [5], [16], [30], [48], [54].

The maximum k-plex on undirected graphs. Many recent works study the maximum k-plex problem [10], [19], [27], [45], [47], [53]. Xiao et al. [47] designed an algorithm with time complexity of $O(c^n n^{O(1)})$ where $c < 2$. Recently, Wang et al. [45] proposed an algorithm which is parameterized by the degeneracy gap (bounded empirically by $O(\log n)$). In addition, a number of techniques were designed to boost the practical performance. Gao et al. [19] developed reduction methods and a dynamic vertex selection strategy. Later, Zhou et al. [53] proposed a stronger reduction method and designed a coloring-based upper bound method for pruning. Jiang et al. [27] proposed a new upper bound method based on the vertex partition while Chang et al. [10] designed an efficient preprocessing step via a comprehensive utilization of the second-order reduction. As mentioned before, all algorithms for the maximum k-plex problem fail to find the maximum DPlex since they ignore the direction information of edges.

Cohesive subgraph models on directed graphs. We summarize several cohesive subgraph models over directed graphs as follows. (1) D-core (or (k, ℓ)-core) [18], [21] is a graph structure where each vertex has at least k in-neighbors and ℓ out-neighbors. (2) D-truss (or (k_c, k_f)-truss) [32], [42] is a concept that counts the triangles that each edge belongs to and splits two different types of triangles in semantics for directed graphs. (3) Densest subgraph [33], [34] refers to a subgraph whose density is the highest. (4) Directed quasi-clique (or (γ_1, γ_2)-quasi-clique) [23] is a type of dense subgraph that requires each vertex has a fraction γ_1 of out-neighbors and a fraction γ_2 of in-neighbors in the subgraph. All the models above define dense subgraphs from different perspectives. However, none of their algorithms can be directly applied to solve our DPlex model.

VII. Conclusion

In this paper, we consider a new directed subgraph model, called directed (k, ℓ)-plex, and study the maximum directed (k, ℓ)-plex search problem. We design a heuristic method DPHeuris, which finds a directed (k, ℓ)-plex with the size close to the maximum one. We then design an exact branch-and-bound method DPBB. Extensive experiments are conducted to show the efficiency of our methods. Next, we will design efficient parallel implementation of our methods.

REFERENCES

[1] R. Albert, H. Jeong, and A.-L. Barabási, "Diameter of the world-wide web," *Nature*, vol. 401, no. 6749, pp. 130–131, 1999.

[2] G. D. Bader and C. W. Hogue, "An automated method for finding molecular complexes in large protein interaction networks," *BMC Bioinformatics*, vol. 4, no. 1, pp. 1–27, 2003.

[3] B. Balasundaram, S. Butenko, and I. V. Hicks, "Clique relaxations in social network analysis: The maximum k-plex problem," *Operations Research*, vol. 59, no. 1, pp. 133–142, 2011.

[4] V. Batagelj and M. Zaveršnik, "An $O(m)$ algorithm for cores decomposition of networks," *CoRR*, vol. cs.DS/0310049, 2003.

[5] R. Behar and S. Cohen, "Finding all maximal connected s-cliques in social networks." in *Proceedings of the International Conference on Extending Database Technology (EDBT)*, 2018, pp. 61–72.

[6] M. Bhattacharyya and S. Bandyopadhyay, "Mining the largest quasi-clique in human protein interactome," in *Proceedings of the International Conference on Adaptive and Intelligent Systems*, 2009, pp. 194–199.

[7] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[8] D. Bu, Y. Zhao, L. Cai, H. Xue, X. Zhu, H. Lu, J. Zhang, S. Sun, L. Ling, N. Zhang *et al.*, "Topological structure analysis of the protein–protein interaction network in budding yeast," *Nucleic Acids Research*, vol. 31, no. 9, pp. 2443–2450, 2003.

[9] L. Chang, "Efficient maximum clique computation over large sparse graphs," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 529–538.

[10] L. Chang, M. Xu, and D. Strash, "Efficient maximum k-plex computation over large sparse graphs," *Proceedings of the VLDB Endowment*, vol. 16, no. 2, pp. 127–139, 2022.

[11] L. Chen, C. Liu, R. Zhou, J. Xu, and J. Li, "Efficient exact algorithms for maximum balanced biclique search in bipartite graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2021, pp. 248–260.

[12] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu, "Efficient core decomposition in massive networks," in *Proc. IEEE Int. Conf. Data Eng. (ICDE)*, 2011, pp. 51–62.

[13] A. Conte, T. De Matteis, D. De Sensi, R. Grossi, A. Marino, and L. Versari, "D2K: Scalable community detection in massive networks via small-diameter k-plexes," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 1272–1281.

[14] Q. Dai, R.-H. Li, M. Liao, and G. Wang, "Maximal defective clique enumeration," *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, vol. 1, no. 1, pp. 1–26, 2023.

[15] Q. Dai, R.-H. Li, X. Ye, M. Liao, W. Zhang, and G. Wang, "Hereditary cohesive subgraphs enumeration on bipartite graphs: The power of pivot-based approaches," *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, vol. 1, no. 2, pp. 1–26, 2023.

[16] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, and X. Lin, "A survey of community search over big graphs," *The VLDB Journal*, vol. 29, pp. 353–392, 2020.

[17] Y. Fang, K. Wang, X. Lin, and W. Zhang, "Cohesive subgraph search over big heterogeneous information networks: Applications, challenges, and solutions," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2021, pp. 2829–2838.

[18] Y. Fang, Z. Wang, R. Cheng, H. Wang, and J. Hu, "Effective and efficient community search over large directed graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 11, pp. 2093–2107, 2018.

[19] J. Gao, J. Chen, M. Yin, R. Chen, and Y. Wang, "An exact algorithm for maximum k-plexes in massive graphs," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2018, pp. 1449–1455.

[20] S. Gao, K. Yu, S. Liu, C. Long, and Z. Qiu, "On searching maximum directed (k,l)-plex (technical report)," https://shuohaogao.github.io/pdf/MaxDirectedPlex-full.pdf, 2023.

[21] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis, "D-cores: measuring collaboration of directed graphs based on degeneracy," *Knowledge and Information Systems*, vol. 35, no. 2, pp. 311–343, 2013.

[22] G. Guo, D. Yan, M. T. Özsu, Z. Jiang, and J. Khalil, "Scalable mining of maximal quasi-cliques: An algorithm-system codesign approach," *Proceedings of the VLDB Endowment*, vol. 14, no. 4, pp. 573–585, 2020.

[23] G. Guo, D. Yan, L. Yuan, J. Khalil, C. Long, Z. Jiang, and Y. Zhou, "Maximal directed quasi-clique mining," in *Proc. IEEE Int. Conf. Data Eng. (ICDE)*, 2022, pp. 1900–1913.

[24] E. Harley, A. Bonner, and N. Goodman, "Uniform integration of genome mapping data using intersection graphs," *Bioinformatics*, vol. 17, no. 6, pp. 487–494, 2001.

[25] S. Janson and M. J. Luczak, "A simple solution to the k-core problem," *Random Structures & Algorithms*, vol. 30, no. 1-2, pp. 50–62, 2007.

[26] A. Java, X. Song, T. Finin, and B. Tseng, "Why we twitter: Understanding microblogging usage and communities," in *Proceedings of the WebKDD and SNA-KDD Workshop on Web Mining and Social Network Analysis*, 2007, pp. 56–65.

[27] H. Jiang, D. Zhu, Z. Xie, S. Yao, and Z.-H. Fu, "A new upper bound based on vertex partitioning for the maximum k-plex problem." in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2021, pp. 1689–1696.

[28] J. Khalil, D. Yan, G. Guo, and L. Yuan, "Parallel mining of large maximal quasi-cliques," *The VLDB Journal*, vol. 31, no. 4, pp. 649–674, 2022.

[29] M. Latapy, "Main-memory triangle computations for very large (sparse (power-law)) graphs," *Theoretical Computer Science*, vol. 407, no. 1-3, pp. 458–473, 2008.

[30] V. E. Lee, N. Ruan, R. Jin, and C. Aggarwal, "A survey of algorithms for dense subgraph discovery," *Managing and Mining Graph Data*, pp. 303–336, 2010.

[31] X. Li, R. Zhou, L. Chen, C. Liu, Q. He, and Y. Yang, "One set to cover all maximal cliques approximately," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2022, pp. 2006–2019.

[32] Q. Liu, M. Zhao, X. Huang, J. Xu, and Y. Gao, "Truss-based community search over large directed graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2020, pp. 2183–2197.

[33] C. Ma, Y. Fang, R. Cheng, L. V. Lakshmanan, W. Zhang, and X. Lin, "Efficient algorithms for densest subgraph discovery on large directed graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2020, pp. 1051–1066.

[34] ——, "On directed densest subgraph discovery," *ACM Transactions on Database Systems (TODS)*, vol. 46, no. 4, pp. 1–45, 2021.

[35] M. Ortmann and U. Brandes, "Triangle listing algorithms: Back from the diversion," in *Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2014, pp. 1–8.

[36] J. Pei, D. Jiang, and A. Zhang, "On mining cross-graph quasi-cliques," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2005, pp. 228–238.

[37] S. B. Seidman, "Clique-like structures in directed networks," *Journal of Social and Biological Structures*, vol. 3, no. 1, pp. 43–54, 1980.

[38] S. B. Seidman and B. L. Foster, "A graph-theoretic generalization of the clique concept," *Journal of Mathematical Sociology*, vol. 6, no. 1, pp. 139–154, 1978.

[39] J. Tang, J. Sun, C. Wang, and Z. Yang, "Social influence analysis in large-scale networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2009, pp. 807–816.

[40] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: Extraction and mining of academic social networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2008, pp. 990–998.

[41] B. K. Tanner, G. Warner, H. Stern, and S. Olechowski, "Koobface: The evolution of the social botnet," in *eCrime Researchers Summit*. IEEE, 2010, pp. 1–10.

[42] A. Tian, A. Zhou, Y. Wang, and L. Chen, "Maximal D-truss search in dynamic directed graphs." *Proceedings of the VLDB Endowment*, vol. 16, no. 9, pp. 2199–2211, 2023.

[43] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theoretical computer science*, vol. 363, no. 1, 2006.

[44] J. Wang and J. Cheng, "Truss decomposition in massive networks," *Proceedings of the VLDB Endowment*, vol. 5, no. 9, p. 812–823, 2012.

[45] Z. Wang, Y. Zhou, C. Luo, and M. Xiao, "A fast maximum k-plex algorithm parameterized by the degeneracy gap," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2023, pp. 5648–5656.

[46] D. Weiss and G. Warner, "Tracking criminals on facebook: A case study from a digital forensics REU program," 2015.

[47] M. Xiao, W. Lin, Y. Dai, and Y. Zeng, "A fast algorithm to compute maximum k-plexes in social network analysis," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 31, no. 1, 2017.

[48] K. Yu and C. Long, "Graph mining meets fake news detection," in *Data Science for Fake News: Surveys and Perspectives*. Springer, 2021, pp. 169–189.

[49] ——, “Maximum k-biplex search on bipartite graphs: A symmetric-bk branching approach,” *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, vol. 1, no. 1, pp. 1–26, 2023.

[50] ——, “Fast maximal quasi-clique enumeration: A pruning and branching co-design approach,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2024, forthcoming.

[51] Z. Zeng, J. Wang, L. Zhou, and G. Karypis, “Coherent closed quasi-clique discovery from large dense graph databases,” in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2006, pp. 797–802.

[52] J. Zheng, M. Jin, Y. Jin, and K. He, “Relaxed graph color bound for the maximum k-plex problem,” *arXiv preprint arXiv:2301.07300*, 2023.

[53] Y. Zhou, S. Hu, M. Xiao, and Z.-H. Fu, “Improving maximum k-plex solver via second-order reduction and graph color bounding,” in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 35, no. 14, 2021, pp. 12 453–12 460.

[54] Y. Zhu, Q. Zhang, L. Qin, L. Chang, and J. X. Yu, “Cohesive subgraph search using keywords in large networks,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 178–191, 2020.

VIII. Appendix

A. Additional Proofs

Theorem 1. *The MDPS Problem is NP-hard.*

Proof. We reduce the decision version of the Maximum k-Plex Search problem (which has been proved to be NP-hard in [3]) to the decision version of the MDPS problem.

Given an undirected graph G, a subgraph $G[S]$ is said to be a k-plex if each vertex connects at least $|S|-k$ vertices in $G[S]$. Then, we are ready to define decision problems used in this proof as follows.

- KPLEX: Given an undirected graph $G=(V,E)$ and one positive integer k, does there exist a k-plex with at least C vertices?
- DPLEX: Given a directed graph $G'=(V',E')$ and two positive integers k' and ℓ', does there exist a directed (k',ℓ')-plex with at least C vertices?

Given an instance I of KPLEX with $G=(V,E)$ and k, we construct an instance I' of DPLEX with $G'=(V',E')$, k' and ℓ'. Specifically, for the directed graph G' in instance I', the vertex set V' is set to be V and the edge set E' consists of directed edges (u,v) and (v,u) if and only if $(u,v)\in E$ in instance I. Moreover, we let $k'=\ell'=k$. It is easy to see that KPLEX is a special case of DPLEX. Therefore, an instance I of KPLEX has a k-plex with at least C vertices if and only if a constructed instance I' of DPLEX has a directed (k',ℓ')-plex with at least C vertices, which completes our proof. □

Lemma 2. *Given a directed graph $G=(V,E)$ and a lower bound lb, a vertex v cannot be included in a directed (k,ℓ)-plex of size greater than lb if $pd_G(v)\le lb$.*

Proof. Based on the definition of pseudo-degree, if there exists a vertex v such that $pd_G(v)\le lb$, we have either $d^{in}_G(v)\le lb-\ell$ or $d^{out}_G(v)\le lb-k$. Thus it is clear that this vertex v cannot be included in a DPlex with size greater than lb. □

Lemma 3. *Given a directed graph $G=(V,E)$ and a lower bound lb, we consider the following two cases according to the relationship between a pair of vertices u and v:*

- *Both edges (u,v) and (v,u) exist in E: if $\Delta^{in}_G(u,v)\le lb-2\ell$ or $\Delta^{out}_G(u,v)\le lb-2k$, then vertices u and v cannot be included in a directed (k,ℓ)-plex of size greater than lb at the same time.*
- *Exactly one of edges (u,v) and (v,u) exists in E: if $\Delta^{in}_G(u,v)\le lb-2\ell+1$ or $\Delta^{out}_G(u,v)\le lb-2k+1$, then vertices u and v cannot be included in a directed (k,ℓ)-plex of size greater than lb at the same time.*

Proof. Assume to the contrary that $(u,v)\in E$, $(v,u)\notin E$, $\Delta^{in}_G(u,v)\le lb-2\ell+1$, and both u and v belong to a DPlex H where $|V(H)|\ge lb+1$. Based on the definition of DPlex, since v is not a in-neighbor of u, there are at most $\ell-2$ vertices that are not in-neighbors of u and at most $\ell-1$ vertices that are not in-neighbors of v in $H\backslash\{u,v\}$. Thus, there are at most $2\ell-3+|\{u,v\}|=2\ell-1$ vertices in H that are not common in-neighbors of u and v. We then conclude that $|V(H)|-\Delta^{in}_H(u,v)\le 2\ell-1$, indicating that $\Delta^{in}_G(u,v)\ge\Delta^{in}_H(u,v)\ge|V(H)|-2\ell+1\ge lb-2\ell+2$, which contradicts our assumption that $\Delta^{in}_G(u,v)\le lb-2\ell+1$. Similarly, we can also obtain a contradiction when $\Delta^{out}_G(u,v)\le lb-2k+1$ and both u and v belong to a DPlex H where $|V(H)|\ge lb+1$. As for the cases of 1) $(v,u)\in E$ and $(u,v)\notin E$ 2) $(u,v)\in E$ and $(v,u)\in E$, the proofs follow a similar fashion. □

Lemma 4. *Given a directed graph G and a directed (k,ℓ)-plex $G[S]$, a vertex $u\in C$ can be removed from G if u has at most $|S|-k$ out-neighbors or at most $|S|-\ell$ in-neighbors in the DPlex $G[S]$.*

Lemma 5. *Given a directed graph G, a directed (k,ℓ)-plex $G[S]$, and a lower bound lb, a vertex $u\in C$ can be removed from C if $pd_{G[S\cup C]}(u)\le lb$.*

Proof of Lemma 4 and Lemma 5. If a vertex $u\in C$ has at most $|S|-k$ out-neighbors or at most $|S|-\ell$ in-neighbors in $G[S]$, then $G[S\cup\{u\}]$ is not a DPlex. Moreover, if a vertex $u\in C$ has $pd_{G[S\cup C]}(u)\le lb$, we know $pd_{G[S\cup\{u\}]}(u)\le lb$ which means that $G[S\cup\{u\}]$ is not a DPlex of size greater than lb by Lemma 2. By the hereditary property of DPlex (Lemma 1), the vertex u can be safely removed from C. □

Lemma 6. *Let $G[S]$ be a directed (k,ℓ)-plex. If we have $|S|>k+\ell-2$, then the diameter of $G[S]$ is no more than 2, i.e., $\forall u,v\in S$, the distance from u to v is at most 2.*

Proof. Assume to the contrary that when $|S|>k+\ell-2$, there exist vertices $u,v\in S$ satisfying the case that the distance from u to v is greater than 2 (or there is no path from u to v). It is easy to see that the four vertex sets u, v, $N^{out}_{G[S]}(u)$, and $N^{in}_{G[S]}(v)$ do not intersect with each other. Further, as $G[S]$ is a directed (k,ℓ)-plex, we know that u has at least $|S|-k$ out-neighbors and v has at least $|S|-\ell$ in-neighbors. Therefore, the four vertex sets in total have at least $1+|S|-k+|S|-\ell+1=2|S|-k-\ell+2$ vertices. In addition, we know all four vertex sets belong to S, which implies that $|S|\ge 2|S|-k-\ell+2$. Thus we can get that $|S|\le k+\ell-2$, which contradicts our assumption and the proof follows. □

Lemma 7. *We are given a directed graph G, a directed (k,ℓ)-plex $G[S]$, a lower bound lb, and $lb>k+\ell-3$. A vertex $u\in C$ can be removed from C if there exist $v\in S$ and $u\notin N^{\le 2}_{G[S\cup C]}(v)$ where $N^{\le 2}_{G[S\cup C]}(v)$ is the set of vertices in $G[S\cup C]$ that have distance at most 2 from the vertex v.*

Proof. By the condition of $lb>k+\ell-3$ and Lemma 6, we know that if there is a larger DPlex H of size at least $lb+1$ in $G[S\cup C]$ that $|V(H)|\ge lb+1>k+\ell-2$ and H includes the vertex set $S\cup\{u\}$, the distance between any two vertices in H is at most 2. Thus the proof follows. □

Lemma 8. *Given a directed graph $G=(V,E)$ and a directed (k,ℓ)-plex $G[S]$ corresponding to the current branch (S,C), let $S=\{v_1,v_2,...,v_{|S|}\}$. Based on the out-neighbors, $\Pi_0,\Pi_1,\Pi_2,...,\Pi_{|S|}$ is a partition of C satisfying:*

- $\Pi_i \cap \Pi_j = \varnothing$, *when* $i \neq j$;
- $\Pi_0 \cup \Pi_1 \cup \Pi_2 \cup ... \cup \Pi_{|S|} = C$;
- $(v_i, u) \notin E$, *for* $1 \leq i \leq |S|$ *and* $\forall u \in \Pi_i$;
- $(v_i, u) \in E$, $\forall u \in \Pi_0$ *and* $\forall v_i \in S$.

The upper bound ub of the maximum directed (k, ℓ)-plex containing the vertex set S can be computed as: $ub = |S| + |\Pi_0| + \sum_{v_i \in S} \min(|\Pi_i|, k - |S \setminus N_G^{out}(v_i)|)$. *Analogously, if C is partitioned into $\Pi_0, \Pi_1, \Pi_2, ..., \Pi_{|S|}$ based on the in-neighbors, we have* $ub = |S| + |\Pi_0| + \sum_{v_i \in S} \min(|\Pi_i|, \ell - |S \setminus N_G^{in}(v_i)|)$.

Proof. We consider the proof for the out-neighbors while the one for the in-neighbors is analogous. The high-level idea to obtain ub is that (1) the candidate set is partitioned into several disjoint sets, and (2) the current directed (k, ℓ)-plex $G[S]$ can only be included a bounded number of vertices in each disjoint set due to the property of DPlex. The proof is then based on these disjoint sets $\Pi_0, \Pi_1, \Pi_2, ..., \Pi_{|S|}$ in the partition. First, since all the vertices in Π_0 are out-neighbors of all the vertices in S, the number of vertices that can be included into the current DPlex $G[S]$ is obviously upper bounded by $|\Pi_0|$. We next focus on Π_i for $i \geq 1$. For each $v_i \in S$, the number of disconnections in S (including v_i itself) is equal to $|S \setminus N_G^{out}(v_i)|$ which implies that the current directed (k, ℓ)-plex $G[S]$ can be included at most $\min\{k - |S \setminus N_G^{out}(v_i)|, |\Pi_i|\}$ vertices from Π_i. This is because a directed (k, ℓ)-plex is only allowed to disconnect at most k out-neighbours. To sum up, the total number of vertices that can be inserted into $G[S]$ from $\Pi_1, ..., \Pi_{|S|}$ is upper bounded by $\sum_{v_i \in S} \min(|\Pi_i|, k - |s \setminus N_G^{out}(v_i)|)$. Thus we complete the proof. □

B. Additional Case Study: Citation Network Prediction

We conduct a case study using our DPlex model to predict the network of citation relationship between authors. To be specific, we download the raw data from AMiner [40] [39] which consists of the information including authors and references of papers from year 2003 to 2018. Then we build the author citation network, where each vertex represents an author, and a directed edge (u, v) is added if there exists at least 5 papers (co-)authored by u that cited another paper (co-)authored by v. For comparison, we use the data before 2013 and before 2018 to build two networks, namely citation-2013 and citation-2018, respectively. We utilize citation-2013 with 1354853 vertices and 3015778 edges to predict the network citation-2018 with 5754125 edges. In particular, we make use of the DPlex model generated in citation-2013 to predict a directed clique in citation-2018, i.e., we aim to find out the potential citation relationship in citation-2018 based on the cohesive subgraph generated from the author citation network in 2013.

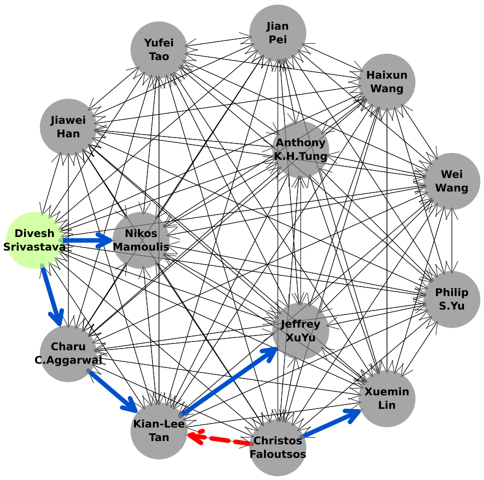
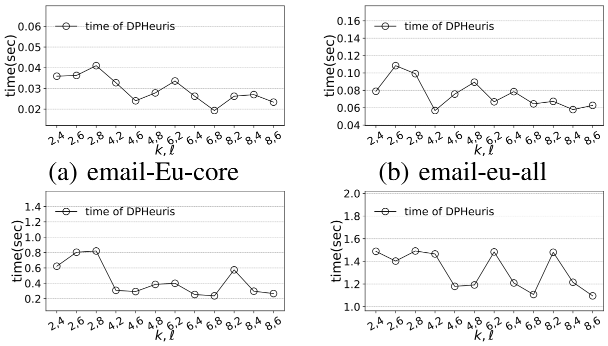


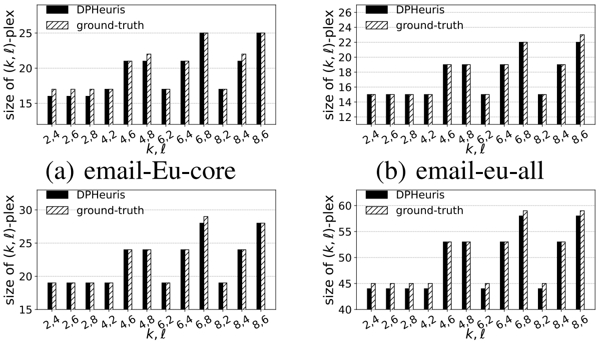
Fig. 14. Citation network prediction using $(3, 3)$-plex



(a) email-Eu-core (b) email-eu-all

(c) soc-Epinions1 (d) imdb-2021

Fig. 15. Time of DPHeuris: Varying k and ℓ



(a) email-Eu-core (b) email-eu-all

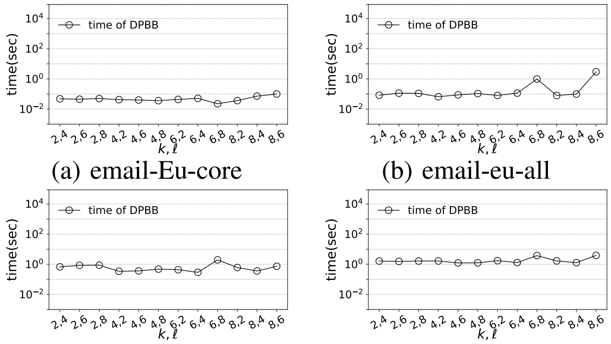
(c) soc-Epinions1 (d) imdb-2021

Fig. 16. Size of DPHeuris: Varying k and ℓ

We select the representing author *Divesh Srivastava* and search the maximum $(3, 3)$-plex containing the vertex associated with *Divesh Srivastava*. As shown in Figure 14, the cohesive structure, e.g., the maximum $(3, 3)$-plex, consists of 14 authors in the area of Data Mining and Database. Then we can predict 6 edges in total and five of them turns out to exist in citation-2018 (the edges that exist in citation-2018 are marked as blue arrows; otherwise, and the other edge is marked as dashed red arrow). This case study shows that our DPlex model is a cohesive structure in directed graphs and can be used to predict the citation relationships between authors.

C. Additional Efficiency Evaluation and Ablation Studies

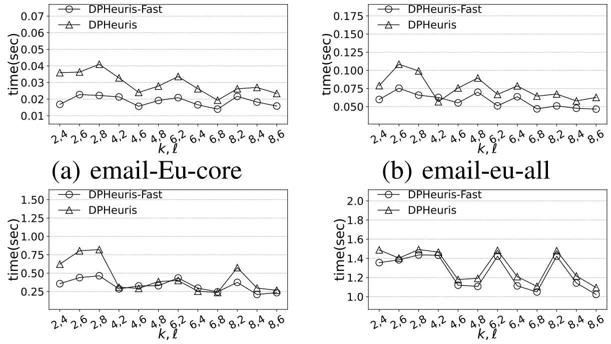
In the following, we report the additional results of our efficiency evaluation and ablation studies with different parameter settings.



(a) email-Eu-core (b) email-eu-all

(c) soc-Epinions1 (d) imdb-2021

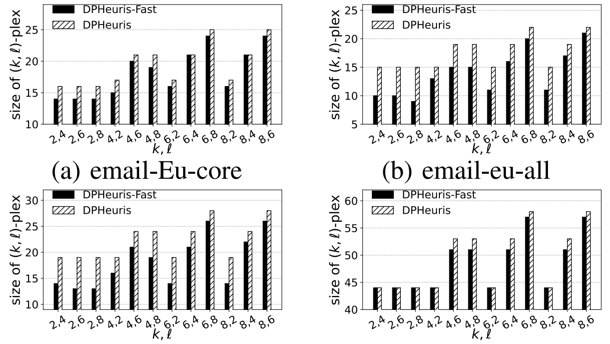
Fig. 17. Time of DPBB: Varying k and ℓ



(a) email-Eu-core (b) email-eu-all

(c) soc-Epinions1 (d) imdb-2021

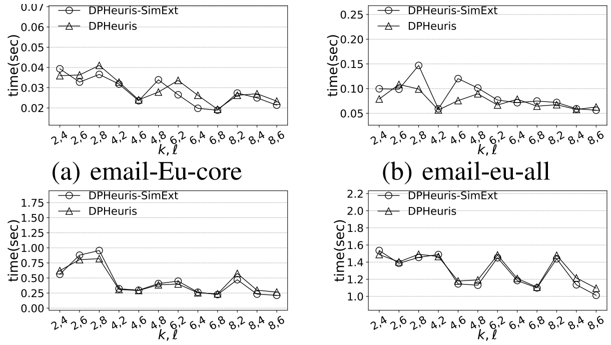
Fig. 18. Time of DPHeuris: Effect of iteration



(a) email-Eu-core (b) email-eu-all

(c) soc-Epinions1 (d) imdb-2021

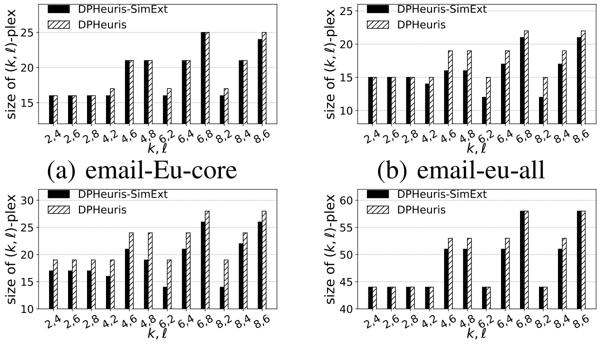
Fig. 19. Size of DPHeuris: Effect of iteration



(a) email-Eu-core (b) email-eu-all

(c) soc-Epinions1 (d) imdb-2021

Fig. 20. Time of DPHeuris: Comparison among various extending strategies



(a) email-Eu-core (b) email-eu-all

(c) soc-Epinions1 (d) imdb-2021

Fig. 21. Size of DPHeuris: Comparison among various extending strategies

DPHeuris: Varying k and ℓ where $k \neq \ell$. We evaluate DPHeuris on default datasets by using unequal k and ℓ, and report the running time and the size of found DPlex in Figure 15 and Figure 16, respectively. We can obtain similar conclusion with that when k is equal to ℓ, i.e., the running time for different values of (k, ℓ) are similar and the sizes of the found DPlex are close to the ground-truth on all settings.

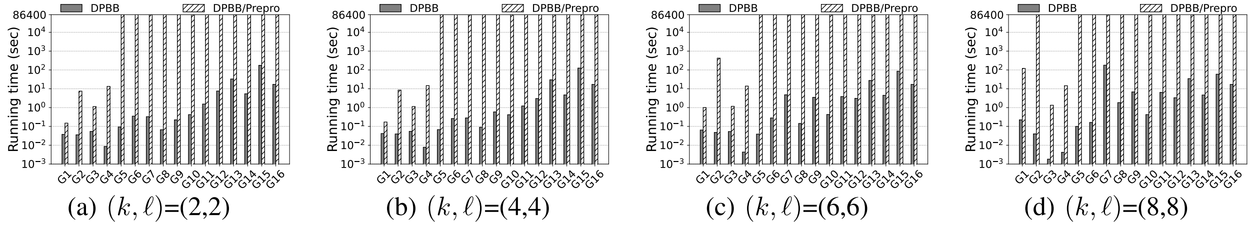
DPBB: Varying k and ℓ where $k \neq \ell$. We evaluate DPBB by varying the values k and ℓ and show the running time in Figure 17. We observe DPBB can handle various settings of k and ℓ, indicating that DPBB is effective and scalable.

DPHeuris: Effect of iteration when $k \neq \ell$. We conduct experiments to compare DPHeuris with its variant DPHeuris-Fast using unequal values of k and ℓ. We report the running time and the size of returned DPlex in Figure 18 and Figure 19, respectively. We observe that DPHeuris runs slightly slower than DPHeuris-Fast and DPHeuris can return larger DPlexes, which is consistent with our findings when k=ℓ.

DPHeuris: Comparison among various extending strategies when $k \neq \ell$. We study the effect of extending strategies in StrongHeuris by comparing DPHeuris with DPHeuris-SimExt. The running time and the size of found DPlex are shown in Figure 20 and Figure 21, respectively. We find that they run comparably fast but DPHeuris can return larger DPlexes, which indicates the effect of our extending strategies in StrongHeuris.

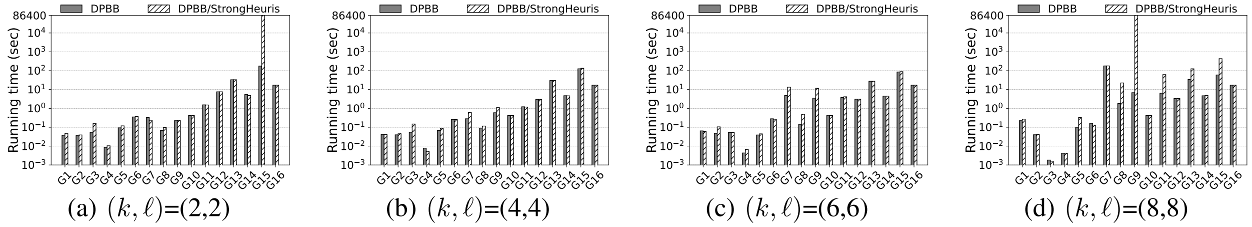
DPBB: Effect of the preprocessing procedure. As show in Figure 22, we study the effect of the preprocessing procedure by comparing DPBB with one variant DPBB/Prepro where (k, ℓ) takes (2,2), (4,4), (6,6) and (8,8), respectively. DPBB can run faster than DPBB/Prepro up to six orders of magnitude, which indicates the importance and effectiveness of the preprocessing procedure.

DPBB: Effect of StrongHeuris. We verify the advance of our StrongHeuris by comparing DPBB with its variant DPBB/StrongHeuris in Figure 23. We can see that our StrongHeuris achieves much better performance especially when (k, ℓ)=(2,2) on G15 and (k, ℓ)=(8,8) on G9. This is because StrongHeuris can provide a larger lb with a small time cost.



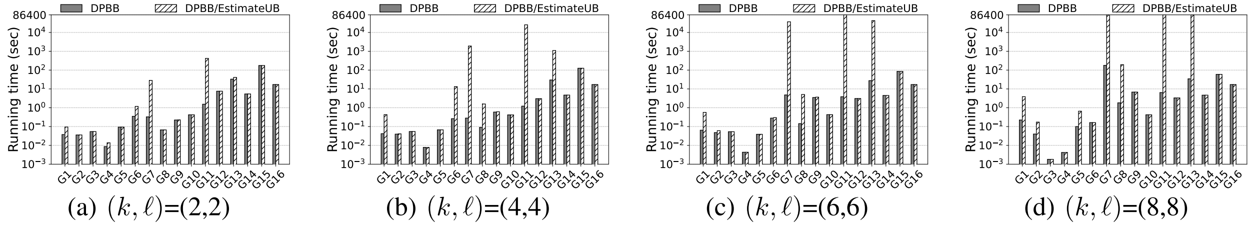
(a) (k,ℓ)=(2,2) (b) (k,ℓ)=(4,4) (c) (k,ℓ)=(6,6) (d) (k,ℓ)=(8,8)

Fig. 22. Time of DPBB: Effect of Preprocessing



(a) (k,ℓ)=(2,2) (b) (k,ℓ)=(4,4) (c) (k,ℓ)=(6,6) (d) (k,ℓ)=(8,8)

Fig. 23. Time of DPBB: Effect of StrongHeuris



(a) (k,ℓ)=(2,2) (b) (k,ℓ)=(4,4) (c) (k,ℓ)=(6,6) (d) (k,ℓ)=(8,8)

Fig. 24. Time of DPBB: Effect of EstimateUB

DPBB: Effect of EstimateUB. We study the effect of the techniques in EstimateUB and compare DPBB with DPBB/EstimateUB, i.e., DPBB excluding EstimateUB in Figure 24. When increasing the values of (k,ℓ), the performances of EstimateUB become more efficient, which indicates the effectiveness of the upper bounding techniques. This is because a larger value of (k,ℓ) will lead to a larger search tree and the pruning techniques become more useful for avoiding unnecessary branches.